Management of the Internet and Complex Services

European Sixth Framework Network of Excellence FP6-2004-IST-026854-NoE

Deliverable D6.2

Open source support & joint software development final report

The EMANICS Consortium

Caisse des Dépôts et Consignations, CDC, France
Institut National de Recherche en Informatique et Automatique, INRIA, France
University of Twente, UT, The Netherlands
Imperial College, IC, UK
International University Bremen, IUB, Germany
KTH Royal Institute of Technology, KTH, Sweden
Oslo University College, HIO, Norway
Universitat Politecnica de Catalunya, UPC, Spain
University of Federal Armed Forces Munich, CETIM, Germany
Poznan Supercomputing and Networking Center, PSNC, Poland
University of Zürich, UniZH, Switzerland
Ludwig-Maximilian University Munich, LMU, Germany
University of Surrey, UniS, UK
University of Pitesti, UniP, Romania

For more information on this document or the EMANICS Project, please contact:

Dr. Olivier Festor
Technopole de Nancy-Brabois — Campus scientifique
615, rue de Jardin Botanique — B.P. 101
F—54600 Villers Les Nancy Cedex
France
Phone: +33 383 59 30 66
Fax: +33 383 41 30 79
E-mail: <olivier.festor@loria.fr>

Document Control

Title:	EMANICS Specification of the static & dynamic content for dissemination environment + Approved and operational Web site
Type:	Public
Editor(s):	Olivier Festor
E-mail:	Olivier.Festor@loria.fr
Author(s):	Mark Burgess, Vincent Cridlig, Olivier Festor, Robert Szuman, Emil Lupu, George Pavlou, Juergen Schoenwaelder, Rolf Stadler, Frédéric Beck
Doc ID:	D6_2-v1_1.doc

AMENDMENT HISTORY

Version	Date	Author	Description/Comments
V0.9	July 9, 2007	Olivier Festor	Start from D6.1 + TOC Extension
V1.0	July 11, 2007	Olivier Festor	Addition of the Chnages & Updates section Addition of the Patterns/CFengine integration section Addition of the NDPMon section Addition of the updated SCLI/Cfengine Integrated section
V1.1	July 24, 2007	Olivier Festor	Addition of the OSIMIS change & update section

Table of Contents

(This page is left blank intentionally.)

Executive Summary

Open Source support and coherent joint software development is a very important initiative for ensuring strong integration, network survivability and transfer of technology developed within the network. In the management community, many Open Source components have been produced over the years, some of them being widely used today for several purposes. Recent evolutions in management and networking technologies however did slow the use of these very valuable software components and the rhythm of new productions has decreased drastically. From the very beginning, EMANICS partners did recognize the need to support the Open Source initiatives in the area of management to foster again their use and acceptance on large scale.

This document is the final report of the activity undertaken during the first 18 months of the network in work-package 6 for increasing the availability, visibility and acceptance of Open Source software in the area of device, network and service management. It describes (1) the efforts made in collecting data on available software, making this data available to the public in an homogeneous and open way and (2) the efforts supported by EMANICS on the development and/or extensions towards broader acceptance and dissemination of Open Source components that emerge from the EMANICS community.

Updates & changes

This document is an updated version of the interim report which was delivered in October 2006. To ease the evaluation of the document by the reviewers, a new section called Updates & Changes has been added to the document. It contains a summary of the changes and extensions brought to the initial document allowing the reviewers to focus only on these changes. The changes are also reflected at the beginning of each section in a short framed text like this one.

0 Updates and Changes

Updates & changes

This section is a new section describing the changes that occur in the revised deliverable of work-package 6 and how they are highlighted in the document.

For the period covered by this update, WP6 activity was centered on finalizing the software developments it did initially support, including those which were started in early December 2007 (coupling of KTH monitoring algorithms with CFengine from HIO). During this period, one additional software development and distribution was granted by the network. This new software is NDPMon an IPv6 Neighbor Discovery Protocol monitoring tool developed by INRIA. SCLI/cfengine integrtion was also achieved during the last 6 months.

Two new section have been added and one section has been completely rewritten and extended in the deliverable to include the description & reporting on the development of these new software components :

- Section 3.5 is a new section that reports the NDPMon activity
- Section 3.6 is a new section that reports the KTH/HIO joint software development initiative
- Section 3.2.4 details the SCLI/cfengine integration. This section is completely new.

Two new appendices were also added: Appendix B is decicated to examples from scli/cfenfine integration. Appendix C describes an execution from cfengine integration with communication patterns from KTH;

Additional changes in the document are minor. They are documented in a framed yellow highlighted text at the beginning of each section (as in this section).

1 Introduction

Updates & changes

The inroduction was updated to include the two additional software components supported by WP6 in the last reporting period. It also reports on the outcome of the last call.

Open Source support and coherent joint software development is a very important initiative for ensuring strong integration, network survivability and transfer of technology developed within the network. In the management community, many Open Source components have been produced over the years, some of them being widely used today for several purposes. Recent evolutions in management and networking technologies however did slow the use of these very valuable software components and the rhythm of new productions has decreased drastically. From the very beginning, EMANICS partners did recognize the need support the Open Source initiatives in the area of management to foster again their use and acceptance on large scale.

The goal of the WP6 work package is to run the tasks related to these Open Source related activities. Its objectives are:

- Promote the use of open source software for device, network and service management purpose,
- Contribute to the identification and evolution of open-source software emerging within the EMANICS network.

To this end, three tasks were identified in Work-Package 6 :

- T6.1: Open source observatory: This task establishes a map of open source software dedicated to network and service management.
- T6.2: Open source repository: This task aims at a classification of high quality open source software developed within MAGIX and its partners.
- T6.3: Software development coordination: This task follows and contributes to a set of sponsored extensions of identified Open source software. Special focus is made on packaging, porting and integration, which are all critical factors to wide adoption.

Two of them were launched after the general assembly in January 2006. These are the Open source observatory task (T6.1) and the Software development coordination task (T6.3).

The design and development of an online observatory was undertaken in task 6.1 in coordination with the simple-web infrastructure. It is described in section 2.

For task 6.3, an open call[3] for Open Source software initatives support was issued to the participants on February 28th, 2006 with closing on March 15th, 2006. Out of the 6 proposals, 4 were selected on March 24th, in year one by the work-package leader together with the executive committee. These initatives are :

- ENSUITE Netconf Extension (INRIA + IUB, 15KEuros support)
- SCLI/CFEngine Integration (HIO+IUB, 15KEuros support)
- OSIMIS Linux Distribution (UniS, 7500 Euros support)
- Ponder Federations and Policy Language Extension (IC, 7500 Euros support)

Two additional packages & joint development activities were supported by WP6 : one still during the first phase but with postponed development start and one after a second call initiated in march 2007. These two components are :

- Patterns in cfengine (HIO+KTH, 15KEuros support)
- NDPMon packaging & distribution (INRIA, 8KEuros support)

A conclusion on the first 18 months of operation of work-package 6 is given in section 4.

[3] The forms to be filled by applicants to this Open Call is provided in appendix A of this deliverable.

2 An online catalogue of available software for network management

Updates & changes

This section was updated (see last paragraph) to report the unification of the database between UT and PSNC.

Create an online catalogue of available software for network and service management arose before the actual start of the project to give users both an easily accessible tool with a user friendly interface and flexible search mechanism to a database of such software in one place – the EMANICS project public Web-site. The initial proposal was to design and implement a map of open-source management software as one tool (activity) and to offer additional visibility to selected software supported by the EMANICS project in a second task. One of the main aims was also to build an improved and best adapted version of existing "online software catalogues".

Various software inventories and lists publicly available in the Internet were evauated and considered to be a base for the software catalogue.

The following examples were found to be the most adequate for our domain and approach :

- the Simple Web "SNMP / Network Management Software" inventory
  http://www.simpleweb.org/software/select_obj.php
- a list of open source software available at
  http://wwwhome.cs.utwente.nl/~fiorezet/ron/Deliverable1.3.1.pdf
- the tool collection at IST MOME website

  http://www.ist-mome.org/database/MeasurementTools/?cmd=toolscategories

Although the above examples together include impressive and huge lists of management tools we have found that they have also had completely different structure of their content and some other flaws (e.g. some links were not up-to-date, some software tools didn't have information about their versions and supported platforms, many tools had only very few descriptive information available, …). During our investigation, we have also found one of the most important disadvantages of the majority of such catalogues – the lack of possibility of editing and updating the existing entries by end-users.

Taking all above arguments into consideration we decided create our own online catalogue of management software which would be free of mentioned flaws and filled with an updated and corrected content. Our catalogue should be also integrated together with the project official web pages and available via the web interface with an advanced search options. Finally to ensure long term survivability of the effort, we decided together with the Simple-Web team to make our databases compatible and maintain their consistency over time.

To fulfil such requirements we have searched and considered many different possibilities of implementation architectures and tools, for example Wikipedia-compatible solutions.

A Wikipedia-like approach seemed to be a basic framework, but unfortunately it is hard in defining data structure/scheme (a kind of web-forms) which can be filled by every partner with detailed description. Also later changes in the structure of Wikipedia are hard to perform (all descriptions have to be changed). It would be also good for our purposes to have two interfaces available: one as read-only for widely open access and second for editing for every partner or for selected and registered users only.

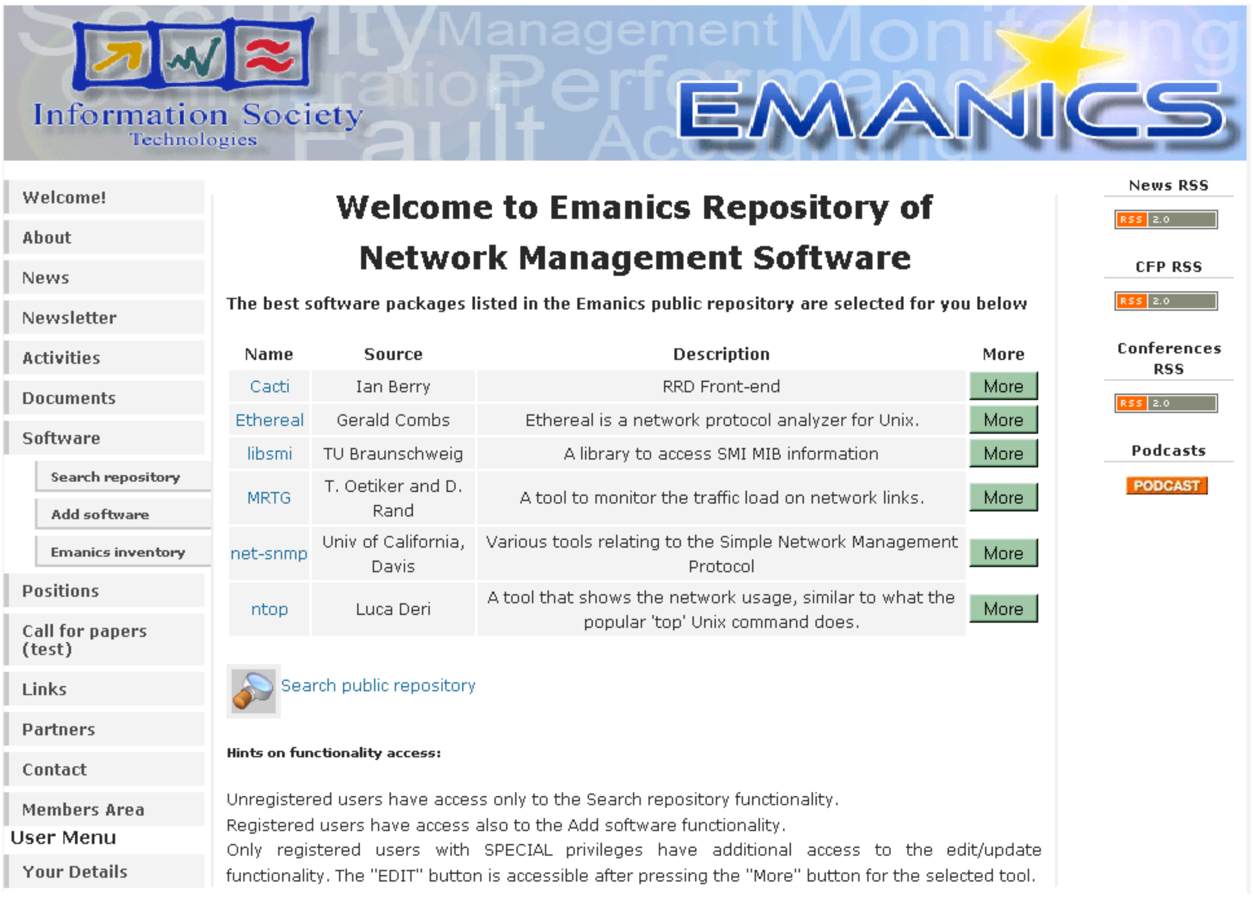
For these reasons we decided to test various modules (components) for the Joomla CMS web-environment which can help to handle with databases and web-form interfaces. Additional benefit of using such Joomla's component is an easy and complete integration with the EMANICS official web pages created also in the Joomla CMS. We have tested several open-source modules of this kind (e.g. MosForms, DBQ) but only "DBQ Manager" was successfully installed and worked properly. Another big problem encountered by us, which is much common to open-source software is the lack of good documentation for the version of tested software and sometimes even the lack of any documentation or tutorial for such type of software. It makes the installation and implementation process of such software much longer and forced us to use a trial and error method during the tests.

Despite many problems encountered while using a free open-source version (1.4) of the "DBQ Manager" component for Joomla CMS, finally also by modifying this component's source code we have achieved all expected milestones shortly described below:

- " EMANICS Repository of Network Management Software" consists of the MySQL database with repository content and configuration tables, the advanced search mechanism with the web-interface accessible to all users (public), the "Add software" functionality accessible for registered users only and the functionality which allows the registered users with special privileges only to edit/update the existing content.
- The table of selected software listed in the EMANICS public repository and evaluated by the SimpleWeb.org and the EMANICS project participants as the best software packages.
- " EMANICS Inventory" consists of the open source software packages selected form the public repository and supported by this project.

All three modules mentioned above are fully integrated with the official EMANICS website and accessible via the "Software" item in the main menu (direct link: http://EMANICS.org/component/option,com_dbquery/Itemid,93/). Their main functionality and available options as well as the short explanations of how to use them are described below together with the screenshots given for better understanding and visualisation purposes.

After successful login to EMANICS pages and choosing the "Software" in the main menu a user can see for example such a page:



Welcome to Emanics Repository of Network Management Software

The best software packages listed in the Emanics public repository are selected for you below

Name	Source	Description	More
Cacti	Ian Berry	RRD Front-end	More
Ethereal	Gerald Combs	Ethereal is a network protocol analyzer for Unix.	More
libsmi	TU Braunschweig	A library to access SMI MIB information	More
MRTG	T. Oetiker and D. Rand	A tool to monitor the traffic load on network links.	More
net-snmp	Univ of California, Davis	Various tools relating to the Simple Network Management Protocol	More
ntop	Luca Deri	A tool that shows the network usage, similar to what the popular 'top' Unix command does.	More

Search public repository

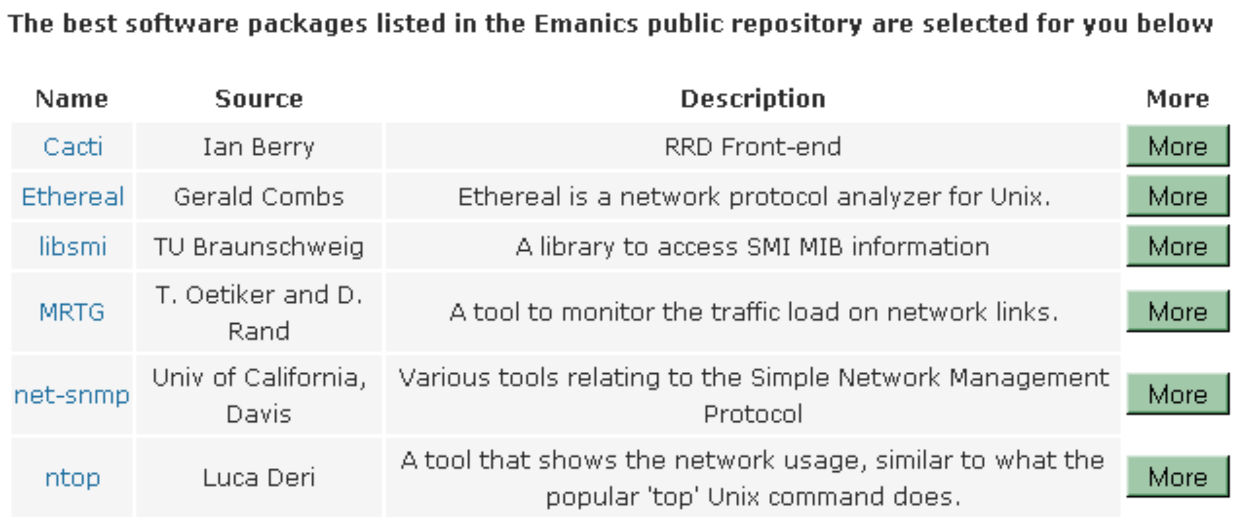
Hints on functionality access:

Unregistered users have access only to the Search repository functionality.
Registered users have access also to the Add software functionality.
Only registered users with SPECIAL privileges have additional access to the edit/update functionality. The "EDIT" button is accessible after pressing the "More" button for the selected tool.

Figure 2.1 The main page of the " EMANICS Repository of Network Management Software".

This main page of the "Software" menu includes three different parts (as shown above):

- the welcome text for EMANICS repository and some hints which shortly explain to the users their accessibility to the various functionality of this repository,
- the table of software packages selected from the EMANICS public repository and recognized as the best and commonly used management tools (see below - Figure 2.2),
- the direct link with icon to the search form of public repository.

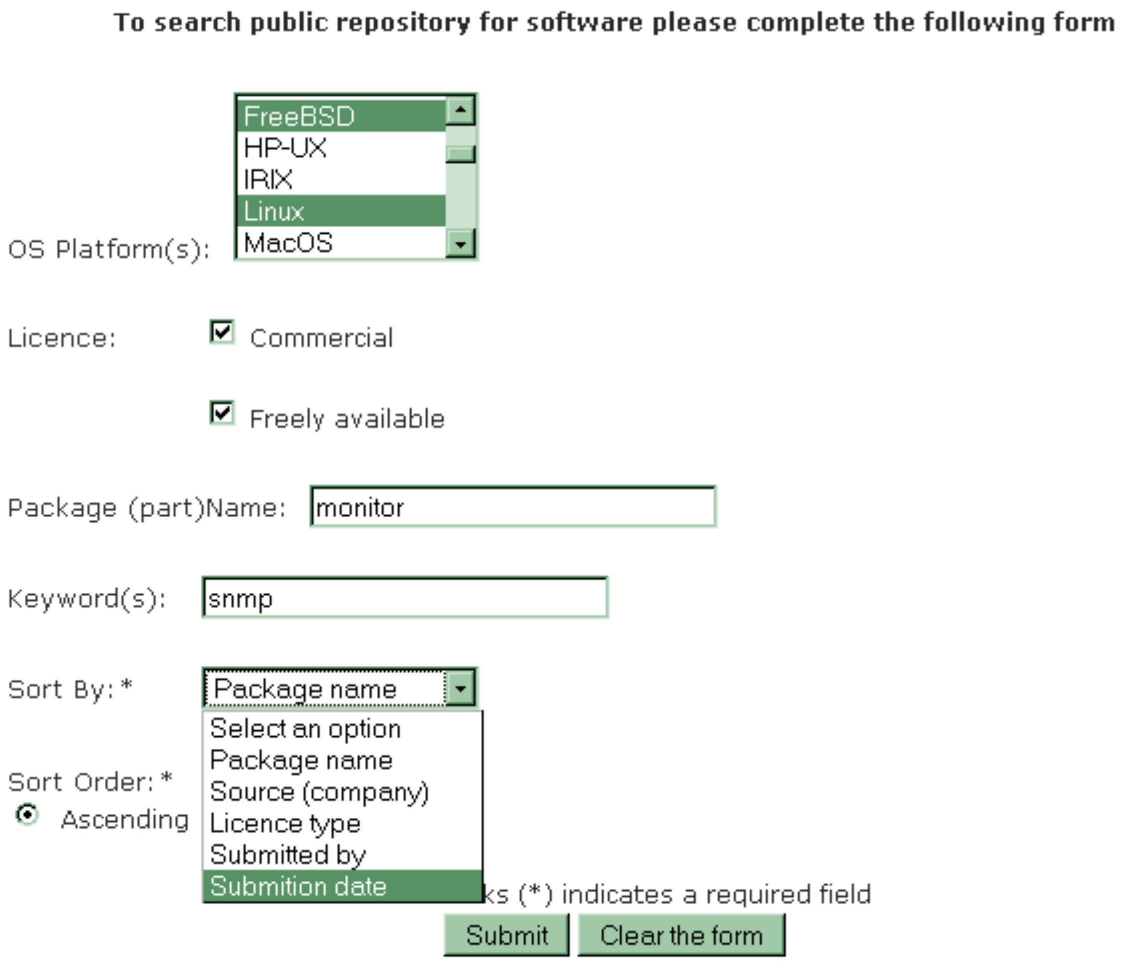


The best software packages listed in the Emanics public repository are selected for you below

Name	Source	Description	More
Cacti	Ian Berry	RRD Front-end	More
Ethereal	Gerald Combs	Ethereal is a network protocol analyzer for Unix.	More
libsmi	TU Braunschweig	A library to access SMI MIB information	More
MRTG	T. Oetiker and D. Rand	A tool to monitor the traffic load on network links.	More
net-snmp	Univ of California, Davis	Various tools relating to the Simple Network Management Protocol	More
ntop	Luca Deri	A tool that shows the network usage, similar to what the popular 'top' Unix command does.	More

Figure 2.1 The table with the best software packages.

In order to find a specific software package(s) there is an advanced searching mechanism implemented and accessible for all users by choosing the “Search repository” submenu item or via the direct link with a magnifying glass icon. It uses a web-form interface to get input data from a user and looks like an example screenshot placed below.



To search public repository for software please complete the following form

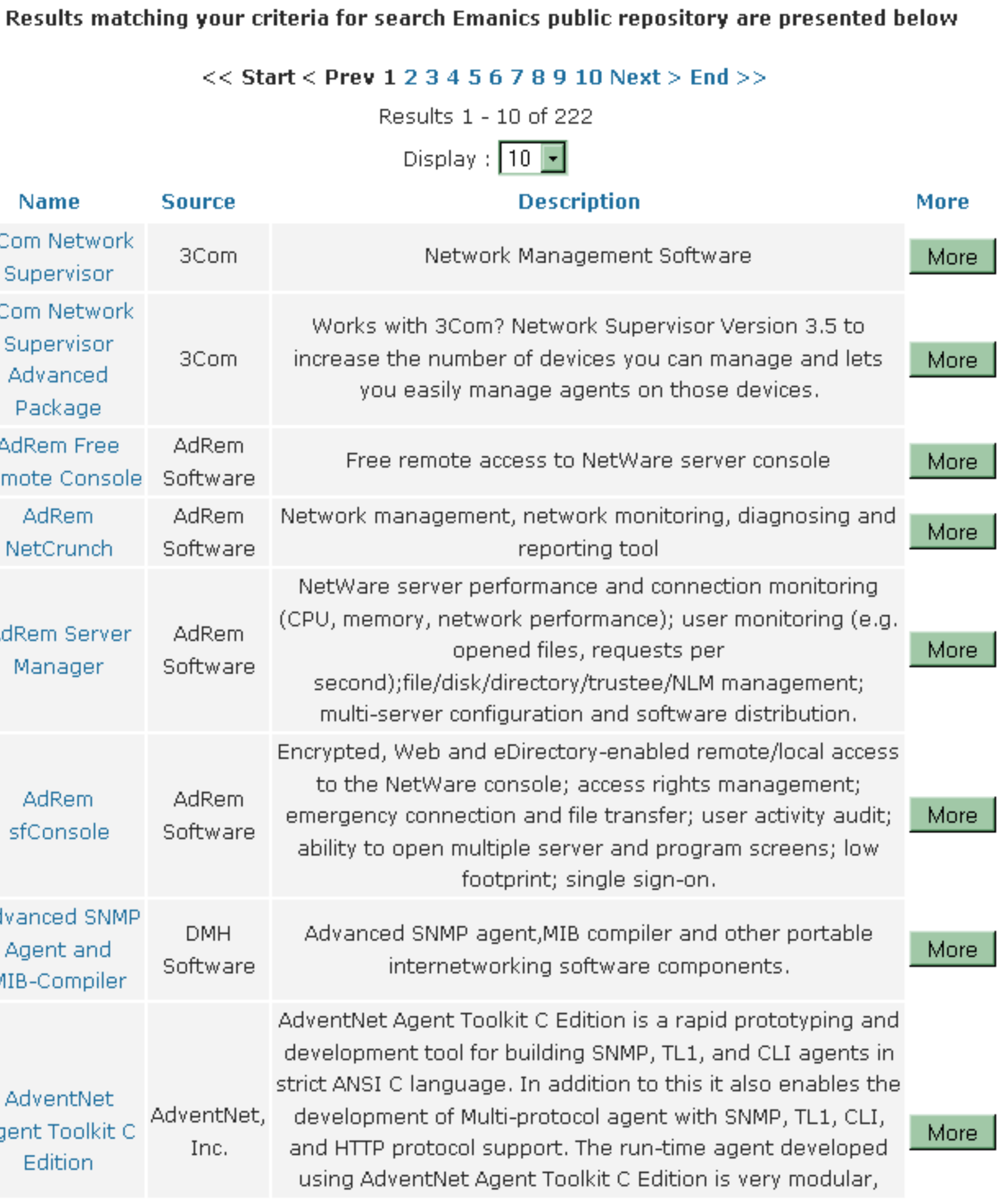
OS Platform(s):	FreeBSD, HP-UX, IRIX, Linux, MacOS (selected: FreeBSD, Linux)
Licence:	[✓ Commercial]
	[✓ Freely available]
Package (part)Name:	monitor
Keyword(s):	snmp
Sort By: *	Package name (options: Select an option, Package name, Source (company), Licence type, Submitted by, Submition date)
Sort Order: *	[● Ascending]

ks (*) indicates a required field

Submit | Clear the form

Figure 2.2 The web-form interface for searching software inside the repository

The results of search functionality are presented as a table on a single web page or several pages with page-navigation links on the top and with sortable columns (by clicking column name). The column called “Name” displays a software package name as an HTTP URL to the product or its vendor home page, if available. Users can also change the number of rows displayed at one web page by changing the “Display” parameter value (HTML form list). Part of the first page of the results for searching for all software included in the repository is shown below as an example.



Results matching your criteria for search Emanics public repository are presented below

<< Start < Prev 1 2 3 4 5 6 7 8 9 10 Next > End >>

Results 1 - 10 of 222

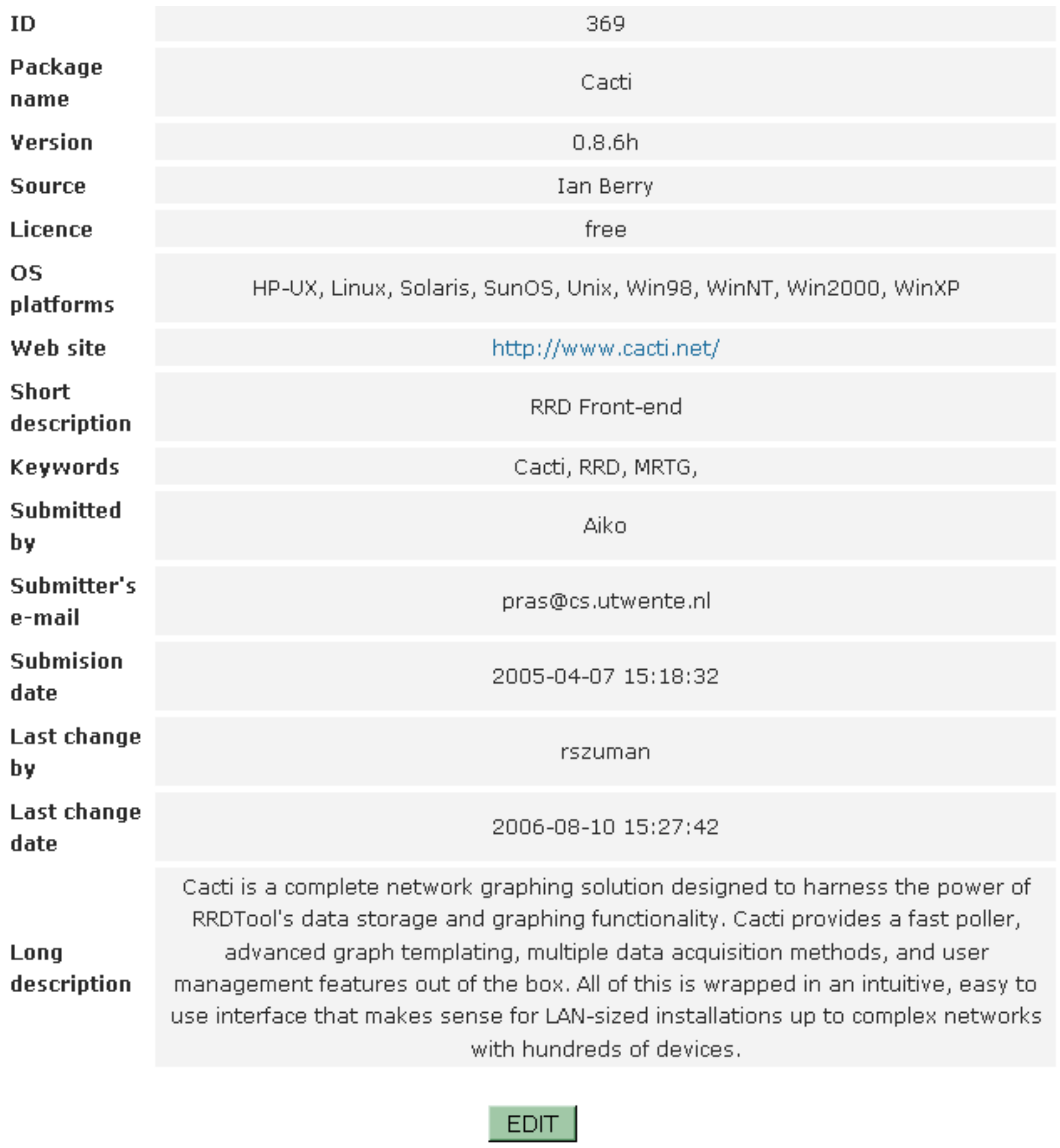
Display : 10

Name	Source	Description	More
3Com Network Supervisor	3Com	Network Management Software	More
3Com Network Supervisor Advanced Package	3Com	Works with 3Com? Network Supervisor Version 3.5 to increase the number of devices you can manage and lets you easily manage agents on those devices.	More
AdRem Free Remote Console	AdRem Software	Free remote access to NetWare server console	More
AdRem NetCrunch	AdRem Software	Network management, network monitoring, diagnosing and reporting tool	More
AdRem Server Manager	AdRem Software	NetWare server performance and connection monitoring (CPU, memory, network performance); user monitoring (e.g. opened files, requests per second);file/disk/directory/trustee/NLM management; multi-server configuration and software distribution.	More
AdRem sfConsole	AdRem Software	Encrypted, Web and eDirectory-enabled remote/local access to the NetWare console; access rights management; emergency connection and file transfer; user activity audit; ability to open multiple server and program screens; low footprint; single sign-on.	More
Advanced SNMP Agent and MIB-Compiler	DMH Software	Advanced SNMP agent,MIB compiler and other portable internetworking software components.	More
AdventNet Agent Toolkit C Edition	AdventNet, Inc.	AdventNet Agent Toolkit C Edition is a rapid prototyping and development tool for building SNMP, TL1, and CLI agents in strict ANSI C language. In addition to this it also enables the development of Multi-protocol agent with SNMP, TL1, CLI, and HTTP protocol support. The run-time agent developed using AdventNet Agent Toolkit C Edition is very modular,	More

Figure 2.3 Part of the results for searching repository functionality.

To see more information about a particular software presented in the general results table (see above), users can click on the package name which is the URL to the product webpage or on the "More" button located on the right side of this software description.

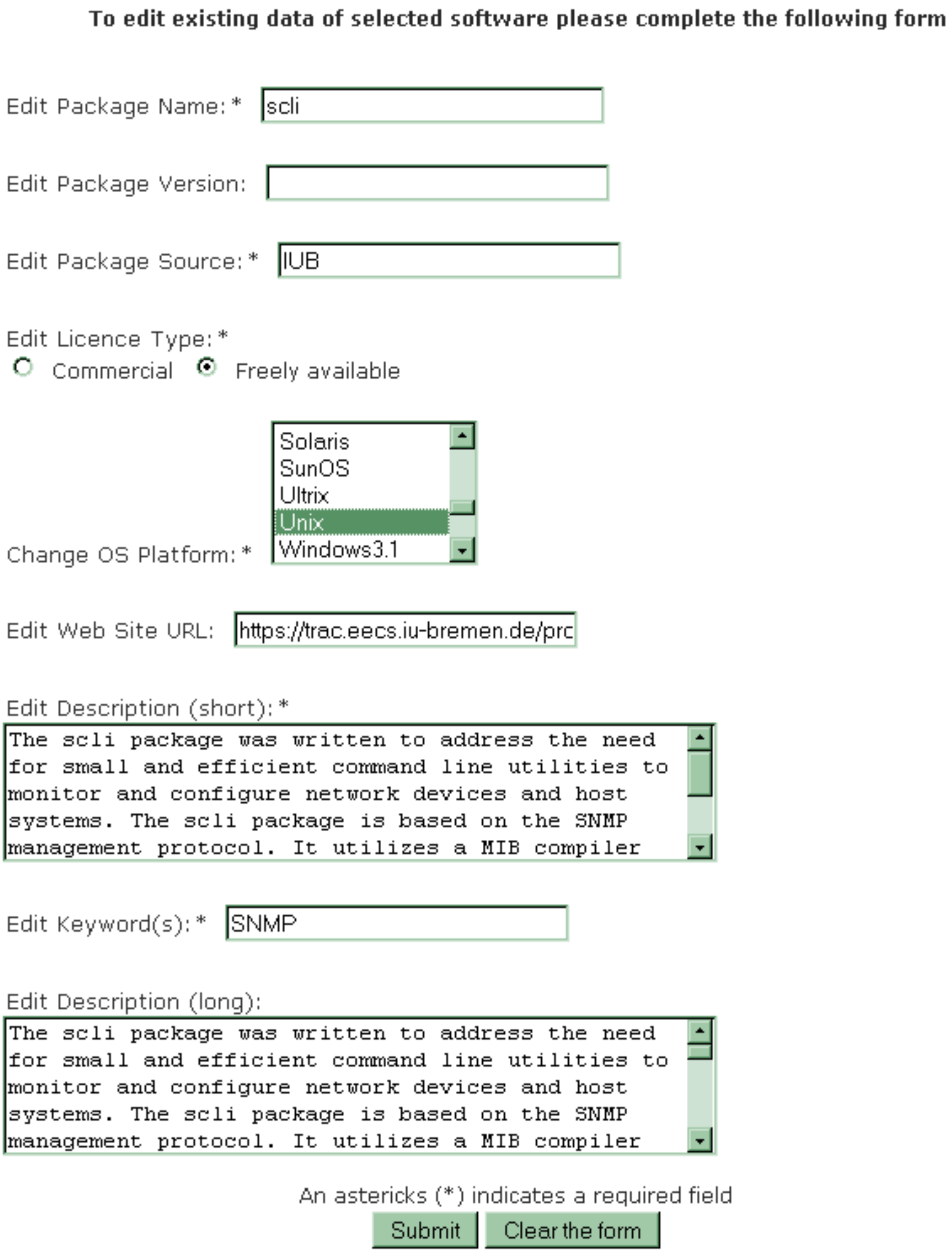
When the second choice is made, by the user, the page with the detailed information about selected software from the EMANICS public repository is presented. It looks analogically to the following example screenshot (the "Cacti" details page).



ID	369
Package name	Cacti
Version	0.8.6h
Source	Ian Berry
Licence	free
OS platforms	HP-UX, Linux, Solaris, SunOS, Unix, Win98, WinNT, Win2000, WinXP
Web site	http://www.cacti.net/
Short description	RRD Front-end
Keywords	Cacti, RRD, MRTG,
Submitted by	Aiko
Submitter's e-mail	pras@cs.utwente.nl
Submision date	2005-04-07 15:18:32
Last change by	rszuman
Last change date	2006-08-10 15:27:42
Long description	Cacti is a complete network graphing solution designed to harness the power of RRDTool's data storage and graphing functionality. Cacti provides a fast poller, advanced graph templating, multiple data acquisition methods, and user management features out of the box. All of this is wrapped in an intuitive, easy to use interface that makes sense for LAN-sized installations up to complex networks with hundreds of devices.

Figure 2.4 The detailed information page about selected software ("Cacti") after choosing the "More" button for it.

All the information available on the details page of any software package can be edited and updated by users with a user-friendly web interface. This functionality is available only for registered users with special privileges in Joomla (e.g. editor rights). When the user with appropriate privileges to update this information is logged in then the “EDIT” button is visible on the bottom of the details page (see example above). To edit the details of selected package, the user has to complete the special form (Pict.2.6).


To edit existing data of selected software please complete the following form

Edit Package Name: * scli

Edit Package Version:

Edit Package Source: * IUB

Edit Licence Type: *
Commercial [✓ Freely available]

Change OS Platform: * Solaris, SunOS, Ultrix, [✓ Unix], Windows3.1

Edit Web Site URL: https://trac.eecs.iu-bremen.de/prc

Edit Description (short): *
The scli package was written to address the need for small and efficient command line utilities to monitor and configure network devices and host systems. The scli package is based on the SNMP management protocol. It utilizes a MIB compiler

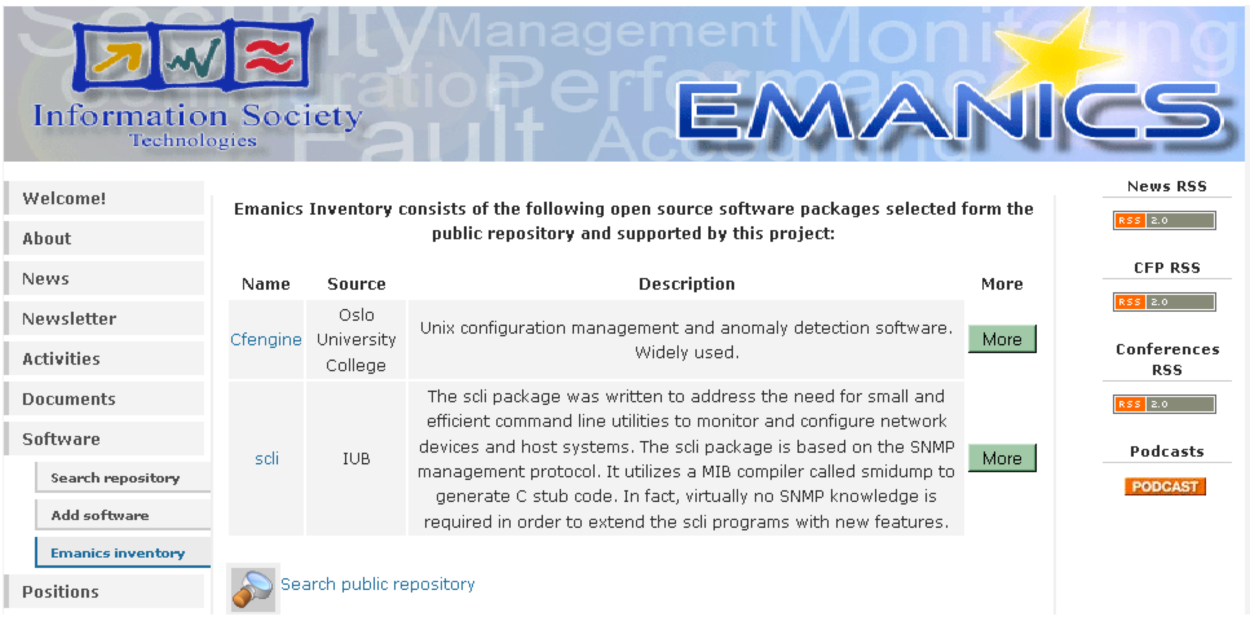
Edit Keyword(s): * SNMP

Edit Description (long):
The scli package was written to address the need for small and efficient command line utilities to monitor and configure network devices and host systems. The scli package is based on the SNMP management protocol. It utilizes a MIB compiler

An astericks (*) indicates a required field

Figure 2.5 The web-form interface for editing/updating detailed information of selected package (“SCLI”).

The last but not the least of the modules mentioned on the beginning of this chapter is the " EMANICS Inventory", which includes open-source software packages supported by the EMANICS project and developed in the scope of it. Current view of this inventory first page is presented below.



Emanics Inventory consists of the following open source software packages selected form the public repository and supported by this project:

Name	Source	Description	More
Cfengine	Oslo University College	Unix configuration management and anomaly detection software. Widely used.	More
scli	IUB	The scli package was written to address the need for small and efficient command line utilities to monitor and configure network devices and host systems. The scli package is based on the SNMP management protocol. It utilizes a MIB compiler called smidump to generate C stub code. In fact, virtually no SNMP knowledge is required in order to extend the scli programs with new features.	More

Search public repository

Figure 2.6 The " EMANICS Inventory" which consists of the software packages officially supported by this project and already included/added in the public repository.

Finally it should be mentioned that the content of the database table with software descriptions used by the inventory is based at the SQL-compatible export of the Simple-Web.org database table. Content of this table was corrected and updated after importing it into the EMANICS repository database, while structure of the table was extended to provide better functionality. The updated and coherent database was then exported from the EMANICS repository and sent back to the SimpleWeb for synchronisation purpose. This cooperation and coordination with the Simple-web will continue for the duration of the network.

In the second reporting of phase 1, it was decided to host the database entirely on the simple-web with access enabled through web services. This solves the database replication and coherence maintenance problem. The implementation of this evolution was done by PSNC together with UT.

3 Supported Open Source Software

3.1 ENSUITE Extensions

Updates & changes

This section was extended to integrate the support of a TR69 gateway within ENSUITE enabling DSL boxes to be managed via Netconf.

The first section does also integrate the move from libresource to sourceforge for the software distribution.

3.1.1 ENSUITE : a Netconf platform

EnSuite is a LGPL implementation of the Netconf configuration protocol in its SSH version. It consists of YencaP Manager which is both a web server and a Netconf manager, and YencaP which is the EnSuite implementation of a Netconf agent. EnSuite is fully developed in pure Python programming language. Documentations, software releases, bugs, forum, news were initially available in the web (http://libresource.inria.fr/projects/ensuite) and integrated in the Libresource environment that is a cooperative platform for software development.

To increase the visibility and acceptance of the software suite, we have moved its distribution to the well known sourceforge software development projects hosting site. ENSUITE is now accessible at (see Figure 3.1.1) :

http://ensuite.sourceforge.net

EnSuite proposes two original capabilities that are typically advertised at session startup. The first one is an access control system, that allows to filter Netconf operations according to the privileges specified in the local agent policy. The second one is a MIB module deployement and management capability which works somewhat like OSGI bundles. In particular, it allows to deploy new extension modules as zip archives into the agent, install and load them dynamically to the program at execution time. This is very useful to update the YencaP software transparently or to deploy new Netconf services at large scale.

EnSuite uses many technologies and standards among XML, XPath, XSLT, DOM, HTML, javascript, CSS, HTTPS, SSH, some of which being dedicated to Netconf, and some others to the web components. EnSuite is very modular, extensible, and well structured thanks to the application of Design Patterns.

The distribution, in terms of lines of code, is as follows:

- Agent: 24 000
  - Core: 11000
  - Modules: 13000 (of which 6000 are from the BGP module and 7 are from the RBAC module)
- Manager: 10 600
  - Core: 9200
  - Modules: 1400

EnSuite is fully compliant with Linux distributions, since it is distributed as:

- rpm package for Fedora Core,
- deb package for Debian and,
- tar.gz for all Linux distributions.

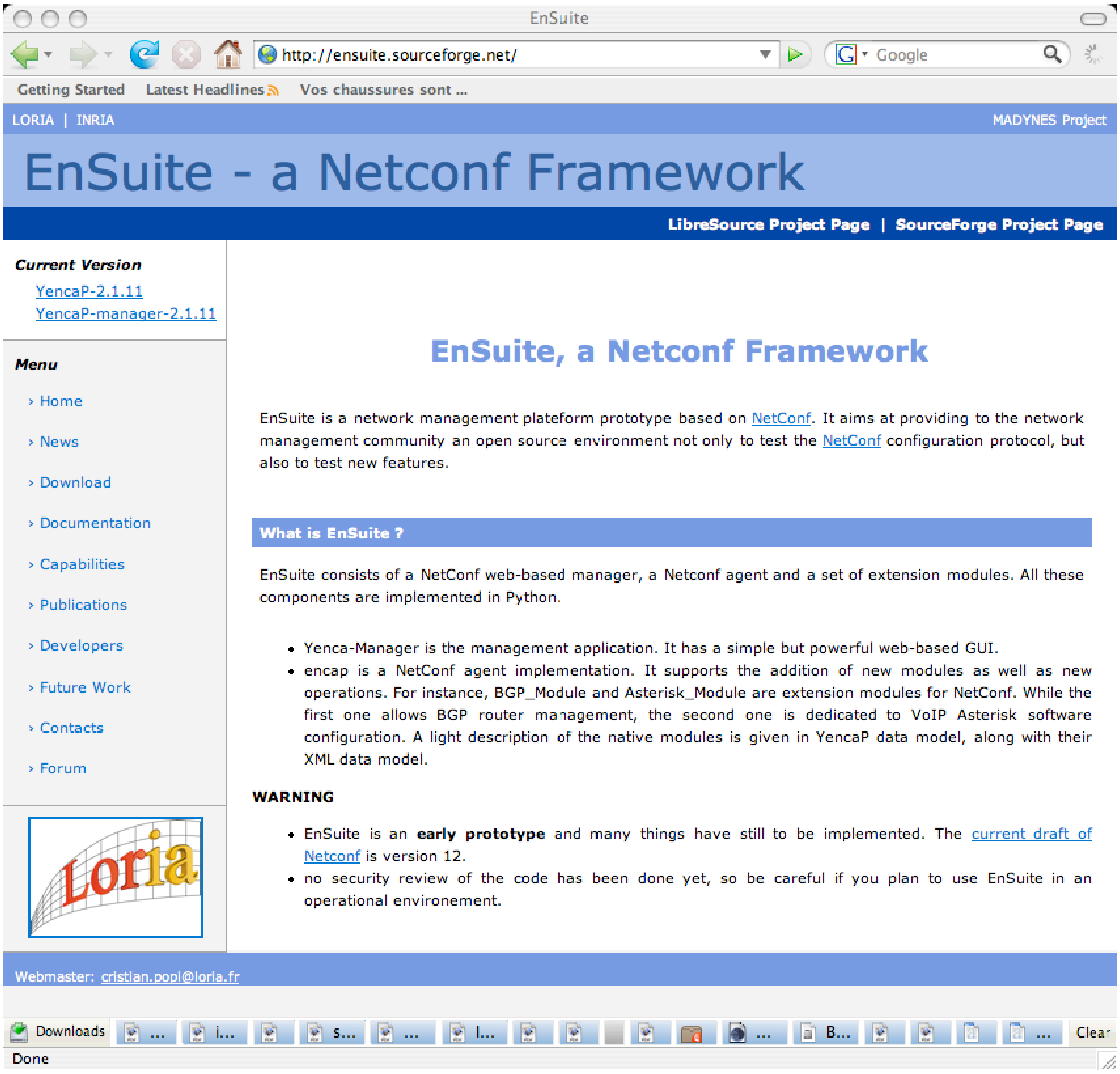
It also supports the classical lifecycle management of Linux services:

- /etc/init.d/yencap [start|stop|restart]
- /etc/init.d/yencap-manager [start|stop|restart]

The dependencies in terms of software packages are as follows:

- python-4Suite-XML-1.0-rc4
- python-amara-1.1.7
- python-paramiko-1.6.1-1
- PyXML-0.8.4

One additional package is required for the agent, quagga-0.98.6-1, and one for the manager, pyOpenSSL.



EnSuite - a Netconf Framework

EnSuite, a Netconf Framework

EnSuite is a network management plateform prototype based on NetConf. It aims at providing to the network management community an open source environment not only to test the NetConf configuration protocol, but also to test new features.

What is EnSuite ?

EnSuite consists of a NetConf web-based manager, a Netconf agent and a set of extension modules. All these components are implemented in Python.

- Yenca-Manager is the management application. It has a simple but powerful web-based GUI.
- encap is a NetConf agent implementation. It supports the addition of new modules as well as new operations. For instance, BGP_Module and Asterisk_Module are extension modules for NetConf. While the first one allows BGP router management, the second one is dedicated to VoIP Asterisk software configuration. A light description of the native modules is given in YencaP data model, along with their XML data model.

WARNING

- EnSuite is an **early prototype** and many things have still to be implemented. The current draft of Netconf is version 12.
- no security review of the code has been done yet, so be careful if you plan to use EnSuite in an operational environement.

Figure 3.1.1: The ENSUITE distribution site

3.1.2 The Yenca agent toolkit

3.1.2.1 General architecture

YencaPs' design follows the layered architecture of the Netconf configuration protocol. It intends to be as modular as possible to allow people to add new capabilities, new operations and extend the data model. To provide this level of modularity, it makes use of some well-known Design Patterns: Command, Singleton, Facade, Composite. YencaP is extensible in three of the four layers of Netconf:

- Server which is the transport protocol layer and can be extended to SSH, BEEP or SOAP or other experimental protocol,
- Operation (or Command) which is extended to Get_Config, Edit_Config, Lock_Config and so on,
- Module which is the Content layer and can be extended to provide new management data.

The major advantage of this flexible architecture is that it allows to add new modules or operations without any change to the code of YencaP core stack. This is not always true but in a majority of case, it is, in particular for modules.

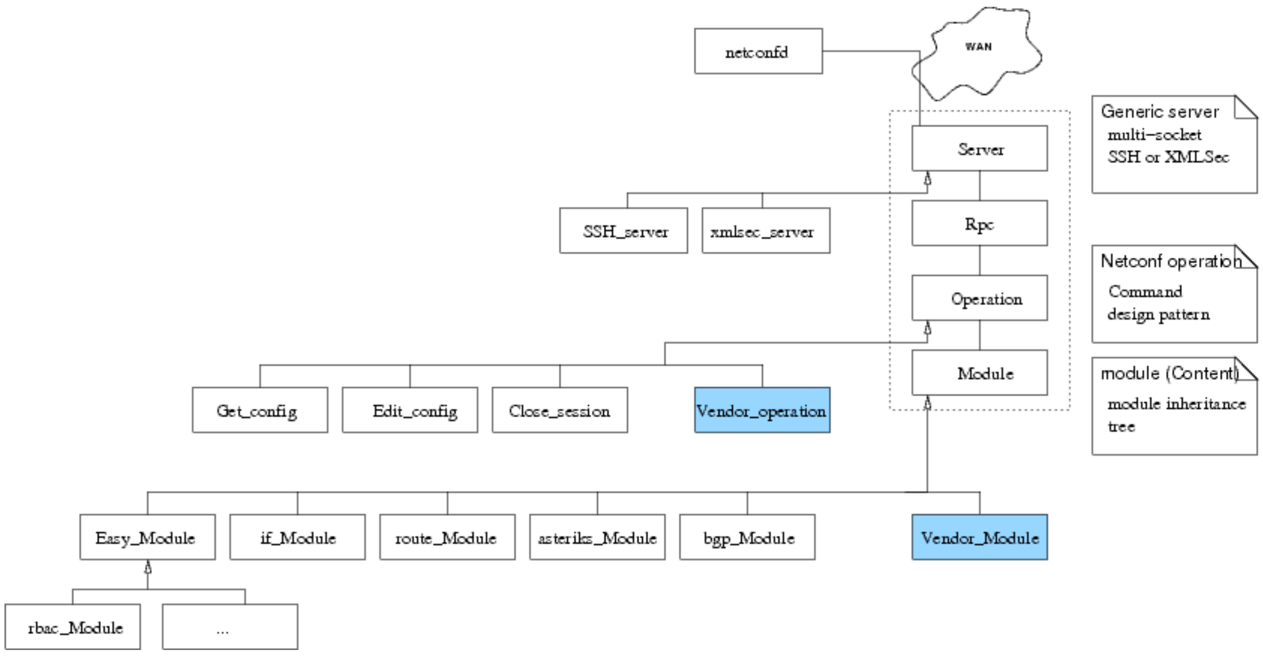


Figure 3.1.2 : The yenca tree

3.1.2.2 YencaP configuration cache

YencaP uses a cache mostly because of the xpath capability. When receiving an XPath request, it is sometimes hard to find the concerned modules because of relative expressions and also the axes which can be very complex (ancestor, childs, and so on). So a cache is maintained according to a cache lifetime specific to each module. A cache makes it very easy to apply XPath requests to the configuration tree. While it consumes more memory, it allows to improve the response time.

3.1.2.3 Layers interfaces

YencaP uses well-defined interfaces to communicate between layers. For example, YencaP defines a ModuleReply class that allows each module to send a standard reply to the Operation layer. The reply can be an error, an positive reply (ok) or configuration data.

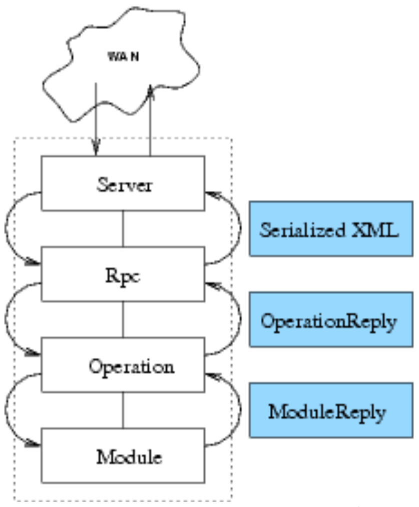


Figure 3.1.3 : Layers in the Yenca framework

3.1.2.4 Operations

Netconf operations are implemented as a Command Design Pattern, which consequently makes the set of operations extensible without changes to the core of YencaP. YencaP has a file descriptor for operations which allows their dynamic instantiation from their names and with a reflection process like in many modern languages. This means that it is possible to implement a new Netconf operation, provide its description and it will work without any change to the code of YencaP.

3.1.2.5 Existing modules

A set of modules is natively included into YencaP. They all inherit from a generic Module class and therefore follow the same API. Consequently, they are all called in the same way and this makes the code of YencaP core identical whatever the module. As for operations, a module can be added without any change to the code. This independance is a major advantage since module developers can benefit from the new versions of YencaP core without problem.

YencaP delegates the management of the different datastores to modules because the way to do it is too dependant on the managed platform (CLI interface, XML data file, Unix pipe, ...). Not all modules support all Netconf operations. For example, IPsec only supports get-config. Here is the list of implemented modules:

- Asterisk Module
- Network Interfaces module
- RBAC module
- Route module
- BGP Module

- RIP module
- TR69-gateway module

Each module defines its own XML data model. This XML subtree is plugged into the Netconf agent XML configuration, which can be seen as a collection of XML subtrees. A XML subtree can be plugged at any level in the XML hierarchy. YencaP defines to sub-classes of Module, namely Easy_Module and CLI_Module. They make its easier to implement new modules by providing a set of methods, for example to connect to the CLI via telnet and send commands or also to autogenerate an XSL file from an edit-config request.

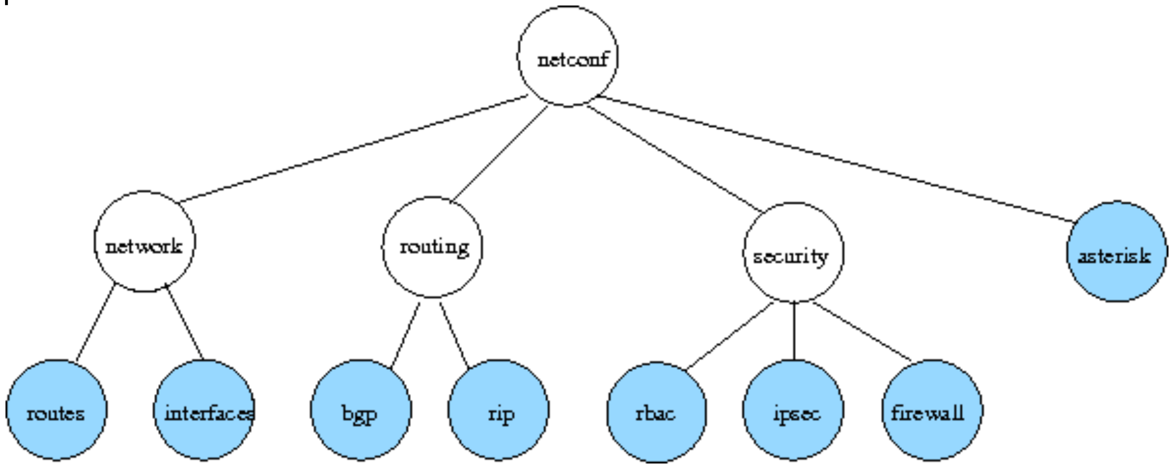


Figure 3.1.4: YENCA available modules

3.1.3 A Netconf manager (YencaP Manager)

YencaPManager is both the Web server and the Netconf manager (client). It implements 95% of the Netconf draft for the manager side. It has basically two main web interfaces: one dedicated to agent selection in case the number of agents is huge, and one dedicated to the management of one agent with all the Netconf operations and the supported modules.

3.1.3.1 General architecture (Composite window)

3.1.3.1.1 Pages hierarchy and composition

A HTML page can be of different types, depending on the navigation context. A page as defined in YencaPManager can be serialized to a string in order to be sent to the web browser of the human manager. We defined three types of Pages:

- MainPage: when a manager is selecting the devices to manage
- AgentConnectedPage: when a manager is connected to (and is managing) a device with Netconf
- AgentReachablePage: when an agent is reachable but is not connected.

AgentConnectedPage itself can be of different types, but they have shared components like for instance, the menu of a connected agent. Therefore the menu is made in the AgentConnectedPage. Here is the list of such pages:

- XpathPage: for sending get-config request based on XPath,
- SubtreePage: for sending get-config request based on subtree filtering,
- LockUnlockPage: for locking and unlocking Netconf devices,

- EditPage: for sending edit-config requests to the Netconf devices,
- and so on… with all Netconf operations.

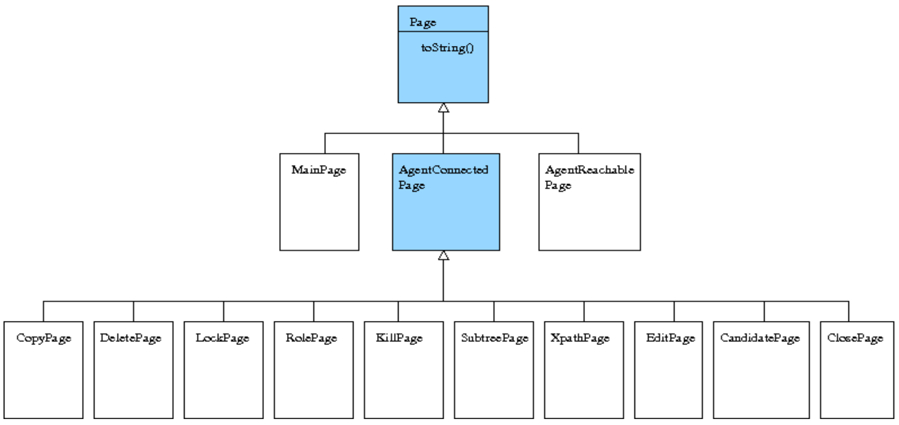


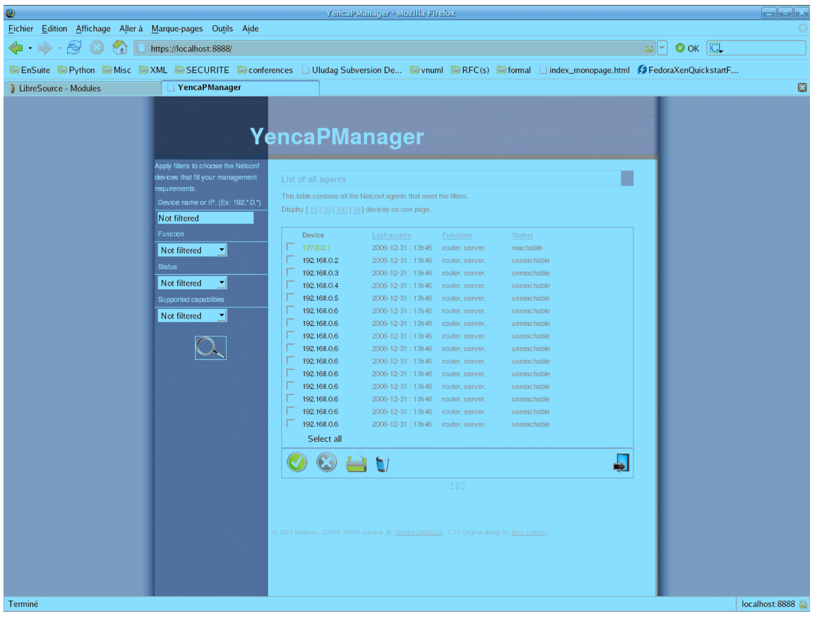
Figure 3.1.5 : Page hierarchy on the YENCA Manager

Like all HTML pages, a Page is basically a composition of components: a header, a content (menu and real content), and a footer. All of them can be in turn compositions of other smaller components, and so on. This is an application of the Composite Design Pattern. The page rendering is managed by a CSS file.

3.1.3.1.2 Agent filtering

The main page of YencaPManager allows to select Netconf agents based on multiple criteria:

- IP address or network
- State (unreachable, reachable, connected)
- Group (server, router, ...)
- Supported capabilities



Apply filters to choose the Netconf devices that fill your management requirements.

Device name or IP. (Ex: 192.*.0.*): Not filtered

Function: Not filtered

Status: Not filtered

Supported capabilities: Not filtered

List of all agents

This table contains all the Netconf agents that meet the filters.

Display [15 | 30 | 100 | All] devices on one page.

Device	Last access	Function	Status
127.0.0.1	2006-12-31 : 13h46	router, server,	reachable
192.168.0.2	2006-12-31 : 13h46	router, server,	unreachable
192.168.0.3	2006-12-31 : 13h46	router, server,	unreachable
192.168.0.4	2006-12-31 : 13h46	router, server,	unreachable
192.168.0.5	2006-12-31 : 13h46	router, server,	unreachable
192.168.0.6	2006-12-31 : 13h46	router, server,	unreachable
192.168.0.6	2006-12-31 : 13h46	router, server,	unreachable
192.168.0.6	2006-12-31 : 13h46	router, server,	unreachable
192.168.0.6	2006-12-31 : 13h46	router, server,	unreachable
192.168.0.6	2006-12-31 : 13h46	router, server,	unreachable
192.168.0.6	2006-12-31 : 13h46	router, server,	unreachable
192.168.0.6	2006-12-31 : 13h46	router, server,	unreachable
192.168.0.6	2006-12-31 : 13h46	router, server,	unreachable
192.168.0.6	2006-12-31 : 13h46	router, server,	unreachable
192.168.0.6	2006-12-31 : 13h46	router, server,	unreachable

Select all

1 2 3

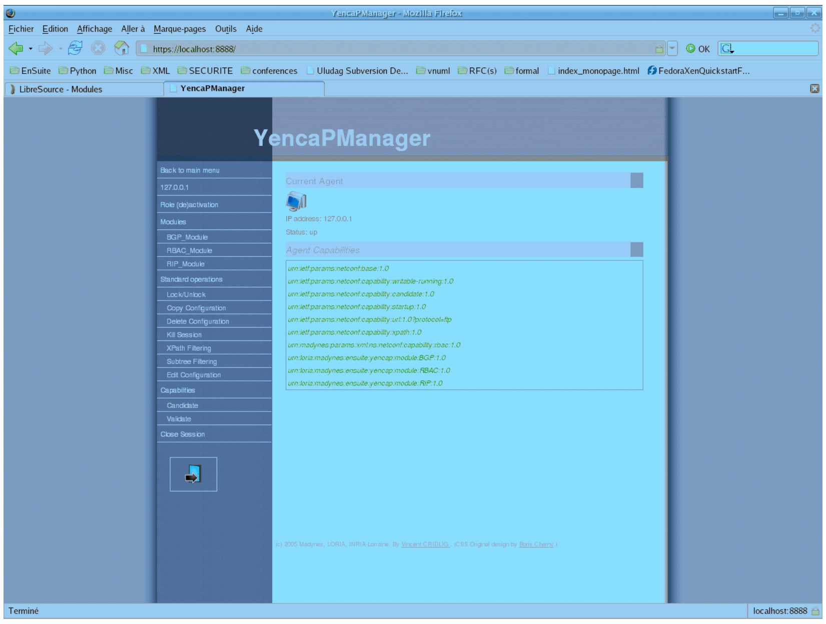
(c) 2005 Madynes, LORIA, INRIA-Lorraine. By Vincent CRIDLIG , (CSS Original design by Boris Cherny.)

Figure 3.1.6: A sampl YENCA Manager page

The agent state is maintained by a separate thread which pings agents periodically. Using a thread improves the response time of the application.

3.1.3.1.3 Netconf Agent page

The Netconf agent page provides two important things: the module forms that allow to update the configuration values and the standard Netconf operations forms. Modules inherit from the Module class. Each module has to provide an XSL file that renders the XML data into HTML code.



Current Agent

IP address: 127.0.0.1

Status: up

Agent Capabilities

urn:ietf:params:netconf:base:1.0
urn:ietf:params:netconf:capability:writable-running:1.0
urn:ietf:params:netconf:capability:candidate:1.0
urn:ietf:params:netconf:capability:startup:1.0
urn:ietf:params:netconf:capability:url:1.0?protocol=ftp
urn:ietf:params:netconf:capability:xpath:1.0
urn:madynes:params:xml:ns:netconf:capability:rbac:1.0
urn:loria:madynes:ensuite:yencap:module:BGP:1.0
urn:loria:madynes:ensuite:yencap:module:RBAC:1.0
urn:loria:madynes:ensuite:yencap:module:RIP:1.0

Figure 3.1.7: A sample YENCA Agent management page

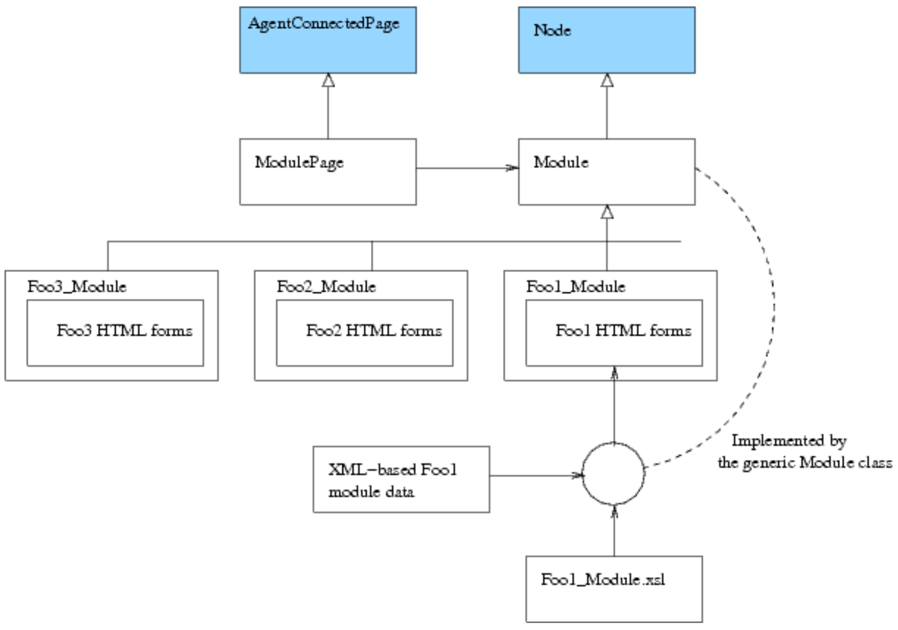


Figure 3.1.8: Agent page structure

3.1.4 Expected impact

NetConf is a new protocol and it remains to be seen how successful it will be in practice. As such, there are many possible long-term scenarios, ranging from NetConf being not adopted as a management protocol (in which case the work would become irrelevant) up to NetConf being highly successful and finally replacing SNMP, in which case the work would be extremly timely and would put EMANICS in the fore front of the technology experts in this are. In other words, there is some risk involved (which is not under control of EMANICS).

EMANICS did clearly benefit by leading implementation efforts in this space and thus gaining influence on the relevant IETF processes (where implementation experience still is considered very important). By improving the NetConf prototype and extending it with NetConf extensions under discussion in the IETF, it will be possible to influence and actively contribute to specifications and standards.

3.1.5 Progress report

Below we provide the progress report in form of a changelog since the beginning of the EMANICS support campaign. The report differenciates the agent (YencaP) and the manager. In these first months, we focused our work mainly on the evolution of the core platform and on its maintenance (task 1).

3.1.5.1 YENCAP

Release 2.1.7

- Access control is done also for copy-config and delete-config
- API changed for Module class
    - def get(self, configName):
- def getConfig(self, configName): where configName is one of "running", "candidate" or "startup". (These constants are accessible via C.RUNNING, C.CANDIDATE, C.STARTUP).
- def copyConfig(self, sourceName, targetName, sourceNode = None):
- def validate(self, targetname, moduleNode = None): where targetname is one of "running", "candidate", "startup" or "config" (the last case requires that the moduleNode must be given as parameter).
- Support validate operation from Maximilian Huetter.

Release 2.1.4

- RBAC improvement for being able to activate a junior role of the assigned roles.
- 4Suite rc4 solved the XPointer bug of the BGP module. Another bug was fixed in BGP module
- Small bug fix in copy-config and edit-config due to namespaces.
- Operations are now loaded by the OperationFactory. Operation information are read from operations.xml by OperationReader. OperationReader now uses amara. OperationFactory now uses the same system as ModuleFactory. OperationManager is the central point for the Operation management.
- Edit-config has been improved. The default operation is first set to the default value. Then, for each module, it checks that the operation attribute is present or not in the parent nodes because this changes the default operation (merge, replace, create, delete). This happens before calling the modules.

- Access control for edit-config has been improved. It selected the authorized nodes in the received request (using XPath scopes). Then if an operation attribute is in the parent nodes then the operation is not allowed since the permissions are granted only on the subtree, not the parents.

Release 2.1.3

- a lot of cleaning (unnecessary imports, XMLSec support)
- continue migration to amara
- lookup tools to build the RPM

Release 2.1.2

- YencaP is now able to upload new modules, install them and load them, all this dynamically.
- split ModuleManager into:
- ModuleManager which is responsible for (un)loading modules on demand. This is implemented as a Singleton,
- ModuleFactory which is responsible for building modules on demand. This is implemented with both a Singleton and Factory design Pattern,
- ModuleReader (formerly ModuleParser) which is responsible for parsing modules on demand (with amara). This is implemented as a Singleton design Pattern.
- Improvement of DatastoreManager code.

Release 2.1.0

- IANA port for Netconf over SSH is now set to 830.

Release 1.1.32

- you don't need to install yapps2, neither to setup the YAPPS environnement variable. We included the two following files to our distrib : yapps2.py and yappsrt.py. This is mostly because yapps2 is not yet available in all the standard Linux distributions as a package. We will consider to change that whenever it is available as an RPM (the deb package is already available).
- XPath request "/" now replies with the full document (if the user has sufficient privileges),
- LogModule now calls the super constructor of EasyModule, not Module,
- The constructor of the modules has been changed and now, each module must call the constructor of the super class Module or EasyModule,
- ...

Release 1.1.25

- Access control is now developed for edit-config. A wiki page is under work to explain the way it works (See YencaP configuration guide),
- Environment variables (YENCAP_HOME...) are no longer required,
- /usr/local/Ensuite/yencap is changed to /usr/local/ensuite/yencap, in order to follow the Linux "best practices",
- /etc/Ensuite/yencap is changed to /etc/ensuite/yencap, in order to follow the Linux "best practices".

Release 1.1.24

- We now provide rpm, deb and tar.gz package.

Release 1.1.23

- Candidate capability is now fully implemented: <commit> and <discard-changes> were missing
- Update <copy-config> according to the new internal design
- Fixes a lot of minor bugs

Release 1.1.22

- fixes a set of bugs in subtree and XPath filtering
- adds support for XML namespaces per module
- Xpath filtering allows the use of prefixes. It is the recommended way.
- moduleResolver has been splitted into two classes: DatastoreManager and ModuleManager.
- DatastoreManager class is responsible for managing the different datastores. In particular, it stores a copy of the running configuration.
- ModuleManager class is responsible for storing the list of modules. It can load, unload and reload modules properly.
- the syntax of the modules.xml file has changed to allow namespace and cache life-time definition

3.1.5.2 YencaPManager

Release 2.1.5

- The page generation process of YencaP Manager was refactored. All pages now inherit from a common model (Page class). This class has three sub classes: mainPage for the agent selection, AgentConnectedPage for managing a given agent and AgentReachablePage which gives minimal info about an agent. AgentConnectedPage has many sub classes corresponding to different things:
  - Forms for Netconf operations
  - Module forms using XSLT

It should be now much easier to build new pages. We also started to update the documentation of YencaP Manager.

Release 2.1.4

- BGP module for yencap Manager side. It allows to define neighbors, route-maps and assignements between them.

Release 2.1.2

- Add support for hot-pluggable modules
- Foo_Module.zip is a sample module that can be used for hotplug demonstration.

Release 2.1.1

- The GUI changed a lot to allow efficient Netconf agent filtering. In particular, in the context of hundreds agents, the GUI makes it possible to select a subset of the agents depending on their status, function, capabilities and IP address.
- IANA port for Netconf over SSH is now set.
- Device status (reachable and unreachable) is now maintained in a separate thread. This improves response time of the application when it must check status of a lot of Netconf agents.
- YencaP Manager now migrates slowly towards python-amara which is really

simple for parsing XML files. *python-amara* is a new package requirement which is distributed in standard FC5 distribution.

Release 2.0.17

- In the web form concerning Subtree and XPath filtering, the prefix list is set automatically accroding to the modules.xml file,

Release 2.0.8

- /usr/local/Ensuite/yencapManager is changed to /usr/local/ensuite/yencap-manager, in order to follow the Linux "best practices".
- /etc/Ensuite/yencapManager is changed to /etc/ensuite/yencap-manager, in order to follow the Linux "best practices".
- Namespaces bugs in RBAC Netconf requests are fixed.

Release 2.0.7

- Add <commit> and <discard-changes> features

3.2 SCLI/CFEngine integration

The objective of this work item is to marry two open source software packages. The first one is cfengine, a policy host configuration system [22]. The second one is scli, a command line interface to network devices running on top of SNMP [23]. In this section, we will describe each package in turn and then outline the work we intend to do, describe the expected impact, and document the progress that has been made so far.

3.2.1 Software description

3.2.1.1 cfengine

Cfengine is a configuration management environment that includes a high level policy language and a low level logic engine. Cfengine ties configuration policy to system monitoring, thereby allowing feedback and reactive configuration. Cfengine does not currently interface with SNMP.

Cfengine can be used as a scripting language for autonomic maintenance of Unix-like computers. For large systems with many different flavours of operating system, what is needed is a disciplined way of making changes which is robust against reinstallation. The idea behind cfengine is to focus upon a few key areas of basic system administration and provide a language which removes decision clutter from scripts, by providing a declarative language, so that the transparency of a configuration program is optimal and management and implementation are kept separate.

Cfengine focusses on a few key functions which are handled rather poorly from scripts. It eliminates the need for lots of tests by allowing you to organize your network according to classes. From a single configuration file (or set of files) you specify, using classes, how your network should be configured -- and cfengine will then parse your file and carry out the instructions, warning or fixing errors as it goes. The basic functions include:

- Checking and configuring the network interface
- Editing textfiles
- Maintaining symbolic links, including multiple links from a single command

- Checking and setting the permissions and ownership of files
- Tidying junk files which clutter the system
- Systematic, automated mounting of filesystems (Unix)
- Checking for the presence of important files and filesystems
- Controlled execution of user scripts and shell commands.

Another component of cfengine performs machine-learning functions. There is no system available in the world today which can claim to detect and classify the functioning state of a computer system. Cfengine incorporates a framework, based on the current state of knowledge, for continuing research into this issue. In version 2.x of cfengine, an extra daemon cfenvd is used to collect statistical data about the recent history of each host (approximately the past two months), and classify it in a way that can be utilized by the cfengine agent.

The current cfengine learning abilities do not extend to SNMP MIBs. By interfacing with scli, it will be possible to incorporate reactive management (so called Event Condition Action behaviour) based on SNMP devices. Communication between cfengine and scli can also allow eventual control of the devices through the cfengine language interface.

3.2.1.2 scli

The program called scli provides an efficient to use command line interface to display, modify and monitor data retrieved from SNMP agents. It runs on simple ASCII terminals and does not require any graphical user interface capabilities. The tool provides command line editing, completion and history capabilities to make it easy for network operators and system administrators to use this tool, even if they cannot remember the precise command syntax.

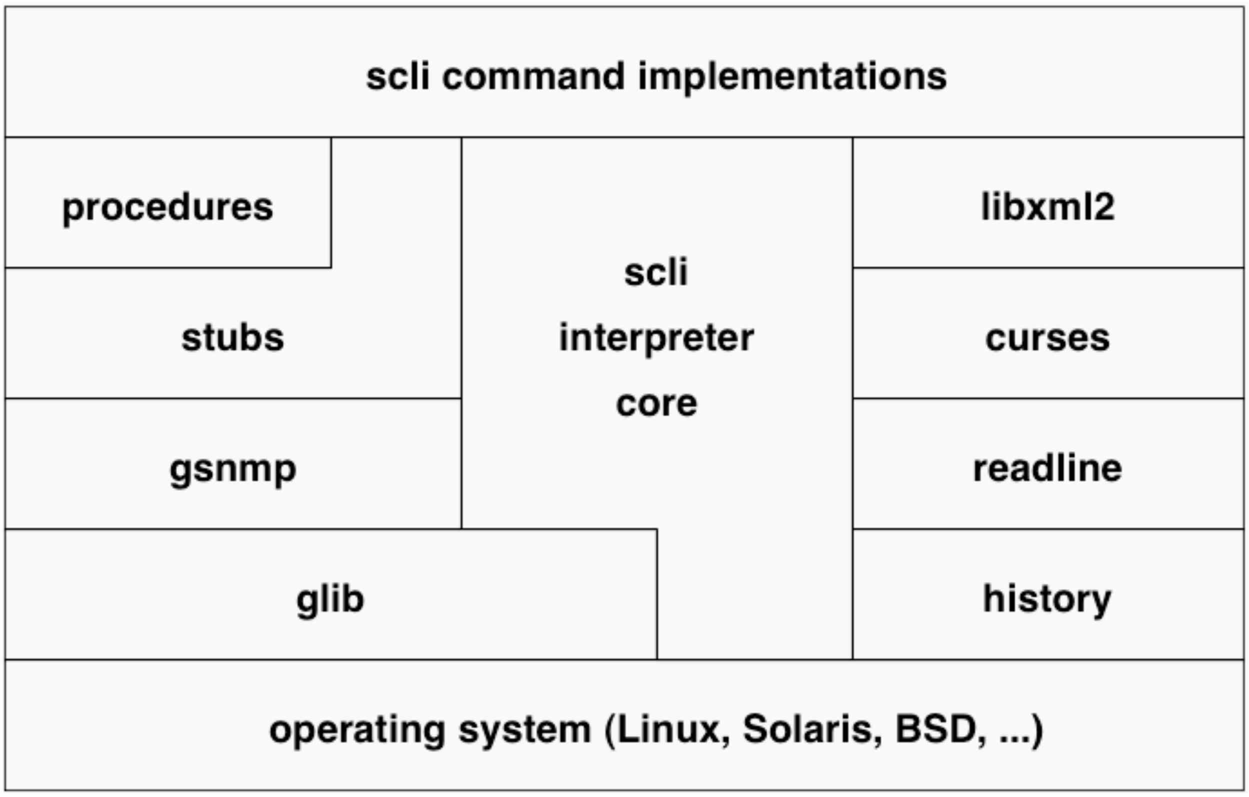


Figure 3.2.1: scli architecture

The overall software architecture is shown in the Figure above. The package uses the glib library to achieve portability and to reuse generic data structures such as lists and dynamic strings. The SNMP engine gsnmp has been derived from the gxsnmp package and was subsequently modified to fix bugs and to improve stability. The SNMP engine itself uses the glib.

The gsnmp library provides roughly the same low-level functionality as many other SNMP APIs. Since it was felt necessary to hide programmers from the low-level SNMP programming details, it was decided to use a MIB compiler to generate C stubs from MIB modules. The stubs consist primarily of C structures which represent MIB table rows or groups of scalars plus a set of stub functions which can be used to read/write these structures. The implementations of the stub functions serialize/deserialize the C structures into SNMP varbind lists. They also validate the data to ensure that the elements of the varbind lists have appropriate types and sizes.

The scli command implementations either use the stubs directly or they use so called MIB procedures. MIB procedures extend the stub interface with specific functions for common operations like creating rows in certain MIB tables or iterating over certain MIB tables. MIB procedures are by definition MIB specific and implemented entirely by using the stub interface.

The scli interpreter core provides all the infrastructure needed to register commands, to tokenize the input stream, to locate and execute the function implementing a recognized command and to finally display the results on stdout or via a pager. The interpreter uses the GNU readline and history libraries for command line editing and the curses library for screen management. It also uses glib data types internally. All state information is bound to the scli interpreter.

The interpreter core and some command implementations also use the libxml2 library to create and manipulate XML documents. By using a dedicated XML library, it is possible to ensure that any generated XML output is well-formed.

3.2.2 Detailed list of extensions planed under the support of EMANICS

Updates & changes

This section has been completely rewritten to report all the developments made in the last 6 months on this integration project.

The main goal is to integrate SNMP access to network devices into cfengine and cfenvd. The approach taken is to use the gsnmp engine and the compiler generated stubs plus a semantic layer which is shared between scli and cfengine. In the end, cfengine should be able to monitor, control and configure remote devices such as layer two switches via SNMP.

Including SNMP data into cfengine will allow monitoring and automated trend analysis of SNMP enabled devices. In the future, it is likely that many, if not all, network devices will run an operating system like Unix/Linux on a small blade card. At such a time, cfengine will be able to run directly on these devices. Today, few systems have this capability, but a remote Linux/Unix machine will be able to query and even configure the devices to a degree, allowing the intelligence to reside in a close-by PC.

3.2.2.1 Monitoring of Parameters using SNMP

The cfengine includes a daemon cfenvd which is collecting longer term statistics in order to detect anomalies. The statistics typically originate from the local host. The goal is to extend this daemon so that it can also collect key parameters from remote systems using SNMP. The approach will be to define an interface over which additional data sources can be integrated with cfenvd. The scli program will be extended to provide such a data source.

Data are read into the daemon and used to update a database using a special algorithm that uses a constant memory size. The daemon automatically adapts to the changing conditions, but has a built-in inertia which prevents anomalous signals from being given too much credence. Persistent changes will gradually change the `normal state' of the host over an interval of a few weeks. Unlike some systems, cfengine's training period never ends. It regards normal behaviour as a relative concept, which has more to do with local stability than global constancy.

The final size of the database is approximately 4MB. Measurements are taken every five minutes (approximately). This interval is based on auto-correlation times measured for networked hosts in practice.

Cfenvd analyses the arrivals and compares them to the learned history. It then sets a number of classes or conditional boolean variables in cfengine which describe the current state of the host in relation to its recent history. The classes describe whether a parameter is above or below its average value, and how far from the average the current value is, in units of the standard-deviation. This can be plotted and analysed further using tools provided by cfengine.

3.2.2.2 Policy Enforcement via SNMP

The cfagent component of cfengine enforces policies written in the cfengine language on the local host system. The goal is to enhance the cfengine language in such a way that it is possible to define policies that should be enforced on network elements that can talk SNMP.

It was agreed to focus on the configuration of VLANs in order to demonstrate the interworking of these software packages. Cfengine implements changes using a method of convergent operations. In recent work, HIO has shown how to implement such operations in a service framework. Here we shall use that framework to implement operations in conjunction with scli. A convergent operator has the property that its repeated application will eventually lead to the base state, and no further activity will be registered thereafter. This requires a slight modification of the operators since the base state must be checked for explicitly.

The implementation makes use of promise theory. The benefits of promise theory are two-fold. Autonomy of the nodes in a promise graph forces us to make all relationships and policy atoms explicit, and the service-nature of the promises makes all changes appear on an equal footing, no matter whether the resources are local or remote, whether the communication is over an internal bus or an external network. This makes certain descriptions that we are used to taking for granted seem cumbersome in promise theory, but it also means that we avoid problems like hidden inconsistencies. Finally, it allows us to take any component of a system and make it into an independently autonomic device.

Using promises we can therefore implement a declarative language interface in cfengine for the SNMP controlled devices. This will require the implementation of a feedback loop through the interface.

3.2.2.3 Concrete Development Steps

The integration of cfengine and scli will be achieved by a loose coupling where a cfengine process may start scli processes in order to collect data or to enforce certain configuration policies. A simple protocol will have to be defined to support the communication between the components. We expect that this protocol will be executed over a local pipe.

Scli will be extended to support additional machine readable output format and a monitoring mode where scli pushes periodically collected statistics to cfenvd. Furthermore, the scli command set will be extended to provide semantics which allow the repeted execution of commands without causing failures (guarded commands).

The cfengine/cfenvd programs will be extended such that they can initiate a pipe to scli and send command and parse the returned output. The details of the protocol between scli and cfengine/cfenvd are still under discussion.

3.2.3 Expected impact

Cfengine as described is used by millions of users worldwide and has an established user base. Scli has been described in a LISA paper butso far is only used by a rather small number of sites. The integration of these two packages will (a) add important features to cfengine in order to configure for example VLAN memberships via cfengine and (b) enhance the technology on which scli is based and at the same time increase its visibility.

Cfengine is published under the GNU GPL2 license. Cfengine is regularly included in Linux and Unix package distributions. Scli is published under GPL, the underlying gsnmp library is under LGPL, and the libsmi package used to generate stub procedures is under a BSD-like license. Scli has been packaged for several Linux distributions including Debian GNU/Linux.

By integrating these two packages, we also integrate software coming from and used by slightly different communities, namely system administrators and network managers. This integration will hopefully open the floor for some synergies and ideally network managers will start to adopt more automated approaches to configuration management while system administrators will take advantage of very lightweight monitoring protocols.

3.2.4 Achieved Integration

This section is completely new and details how integration has been achieved.

The integration of `cfengine` [22] and `scli` [23] has been achieved by a loose coupling where a `cfengine` process may start scliprocesses in order to collect data or to enforce certain configuration policies. A communication channel will be used between `cfengine` and `scli` in order to send and receive messages. The `gsnmp` engine and the compiler generated stubs of `scli` will be used to talk SNMP to remote devices. Thus, `scli` will serve as a proxy between cfengineand the remotely managed device. In order to achieve the goals several task have to be carried out:

- A communication protocol between `cfengine` and `scli` has been defined. This protocol defines the syntax of the messages that are exchanged between `cfengine` and `scli`.
- The `scli` implementation has been adapted so that it can handle multiple requests concurrently. The most convenient way of doing that was to enable thread support for `scli`. Hence the `scli` code has been adapted so that it is thread-safe and the functions are re-entrant. This extension allows a single `scli` request to handle various requests concurrently by a separate synchronized threads.
- The `cfengine/cfenvd` programs has been extended such that they can initiate a pipe to `scli` and send commands and parse the returned output.

3.2.4.1 Scli Protocol

The `scli` protocol is a request/response protocol which provides a way for an external program like `cfengine` to read data from managed devices and to configure them by utilizing simple command messages.

3.2.4.1.1 Protocol Specification

An `scli` message is either a request or a response message. A request message is sent to an `scli` interpreter. `scli` has one master interpreter and can have many slave interpreters. A request message sent to the master interpreter consists of a number of tokens. Each token can contain letters, digits and special characters excluding single and double quotes and the `#` character. Tokens themselves can be encapsulated into single or double quotes. A request sent to a slave interpreter is always prefixed with the name of the slave interpreter. Interpreter names can contain letters, digits as well as dots and colons and must always start with a letter. All requests sent to `an scli` interpreter must end with a carriage return and line feed (CRLF).

A response message consists of a message from the master interpreter and can be optionally followed by a message from a slave interpreter. A message from the master interpreter contains a response code followed by a short description of the code and it can optionally carry data. Whenever a response contains data each line of data is prefixed with a response code followed by a "-". The last line of the master interpreter response contains the final response code together with a short description. A message from a slave interpreter might follow the message from the master interpreter (in the case when the request message was sent to the slave interpreter). The response from the slave interpreter has the same structure as the response from the master interpreter, however each response line is prefixed with the name of the slave interpreter. All lines in the response end with a carriage return and line feed (CRLF).

The ABNF specification of the scli protocol is shown in the Appendix B .

3.2.4.1.2 Response Codes

Response codes can be classified into one of the following categories:

- 1xy - transient messages
- 2xy - positive completion
- 3xy - generic error codes

- 4xy - errors detected before command processing (syntax errors)
- 5xy - communication failure

3.2.4.1.3 Communication Architecture

An external program establishes a communication channel with an `scli` interpreter and sends commands, subsequently called request messages. The `scli` interpreter generates SNMP messages based on the data contained in the scli request messages and sends them to the respective managed device. When SNMP responses are received, the data is converted in an `scli` format and sent back to the external program as an `scli` response message. The communication between `cfengine` and `scli` is shown in Figure 3.2.2 below.

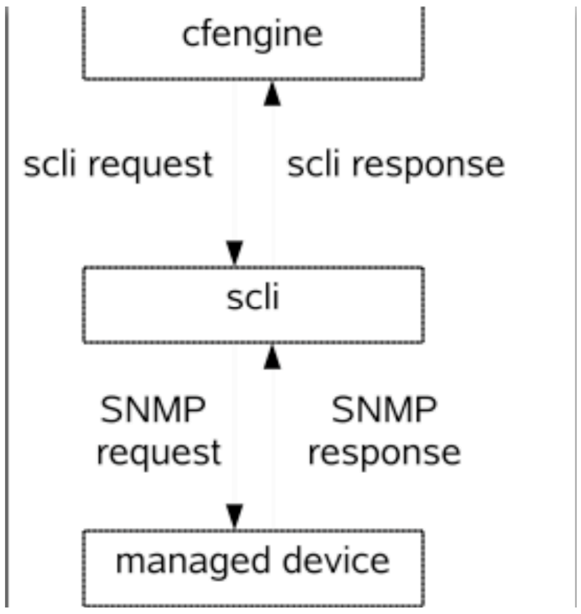


Figure 3.2.2: Communication between cfengine and scli

An entity which runs `scli` contains a master interpreter and can optionally contain several slave interpreters. When a request message is sent via an I/O stream to the entity running `scli`, the master interpreter handles the message. If the command inside the request message is destined for the master interpreter, then it executes the command. If the command inside the message is destined for a slave interpreter, the master interpreter communicates the command to the respective slave interpreter via an `scli` internal communication mechanism after checking if the respective slave interpreter exists. A scenario of `scli` with a master interpreter and three slave interpreters each one serving a different SNMP agent is shown in Figure 3.2.3.

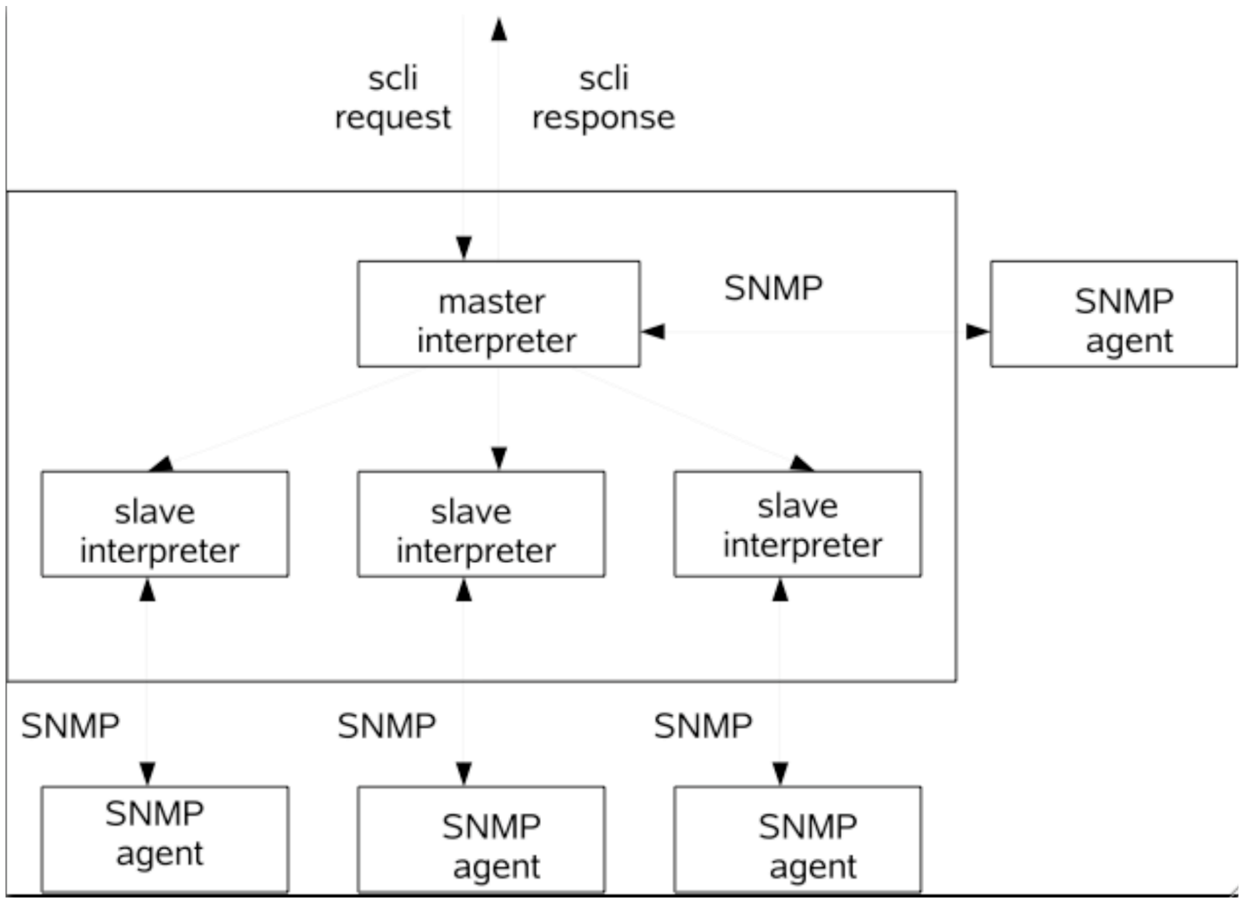


Figure 3.2.3: `scli` master interpreter and several slave interpreters

3.2.4.2 Elements of Procedure

The master `scli` interpreter waits for request messages (commands). The set of specific commands as well as the syntax of the commands is specified in the `scli` documentation[4] Whenever a slave interpreter is created it registers its identifier as a command with the master interpreter. It is the responsibility of the external program that invokes the `scli` interpreter to ensure that the syntax of the command invoked is correct.

Whenever a command is received by an interpreter it is initially parsed and the tokens are extracted. If tokenizing fails, then the error code 405 (syntax error in tokenizer) is returned by the respective interpreter. If tokenizing is successful then the command is matched against the set of registered commands. If there is no such command registered, then the error code 406 (syntax error unknown command) is returned by the interpreter. At this point if the command is destined for a slave interpreter of the current interpreter the command as well as the arguments are communicated to the respective slave interpreter and the current interpreter returns a status code of 200 (ok) thus declaring that it is done with processing the command. Any error or status codes as well as data that results from the execution of the command by the slave interpreter will be returned by the slave interpreter. If the command requires communication with an

[4] https://trac.eecs.iu-bremen.de/projects/scli/

SNMP agent the scli interpreter checks whether the interpreter is associated with an SNMP agent and whether the output format requested is supported by the command. If the command requires and SNMP association and there is no SNMP agent associated with the `scli` interpreter executing the command, then the error code 301 (no association to SNMP peer) is returned by the respective interpreter. Furthermore, if the output is requested in XML format and the command does not support XML output format, then error code 302 (command does not support XML) is returned by the interpreter. Otherwise, the interpreter checks the number of arguments provided. If a wrong number of arguments is provided, then error code 401 (syntax error in number of arguments) is returned. After verifying that the correct number of arguments is provided, each argument is checked whether it belongs to the correct type and range. If the check fails, then one of the syntax error codes is returned by the interpreter: error 402 (syntax error in regexp), error 403 (syntax error in number) or error 404 (syntax error in value). If any other syntax error is detected, then error 400 (generic syntax error) is returned. If all checks are successful the respective interpreter starts executing the command. If there is a runtime error during the execution of the command, then a respective error code is returned by the interpreter executing the command. The runtime error codes are: error 500 (snmp error return code) or error 501 (snmp name lookup error). If the command has been successfully executed, then the respective `scli` interpreter returns the data (if any) together with a status code of 200 (ok).

The processing of commands by scli interpreters is described in Figure 3.2.3. Some example sessions of using the scli protocol mode of operation are shown in the Appendix B.

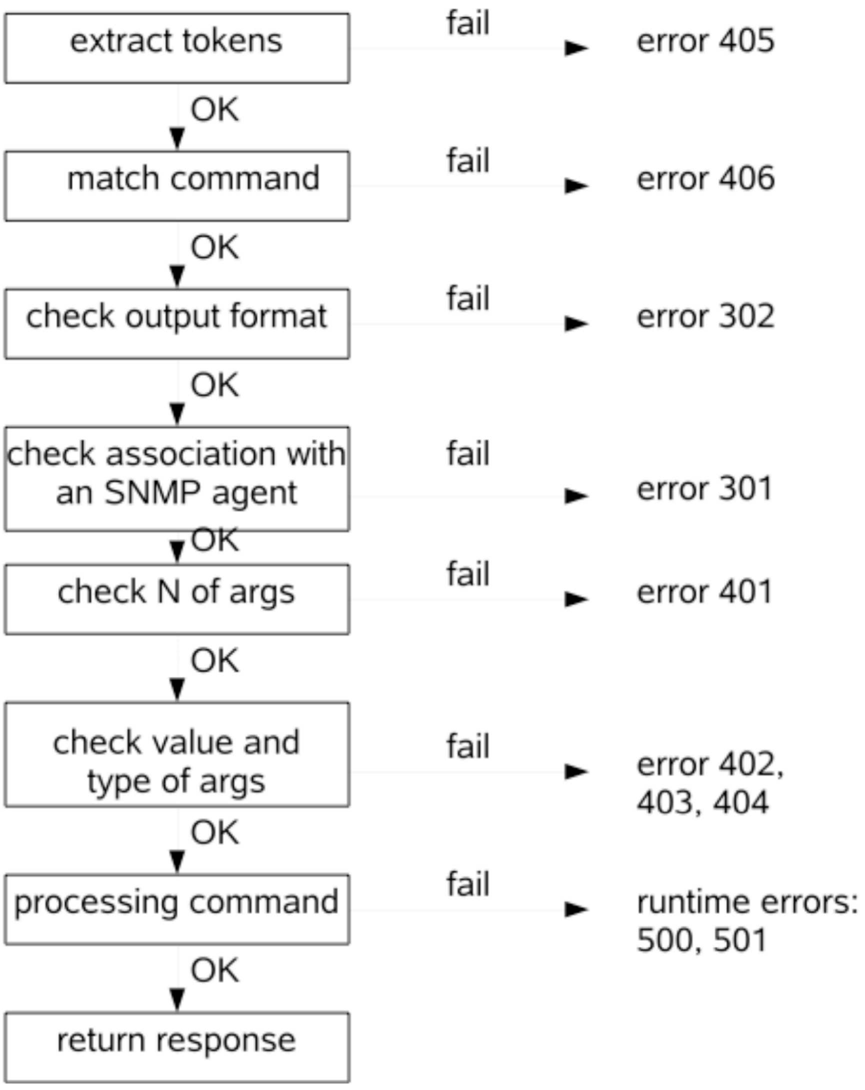


Figure 3.2.4: Processing of commands by `scli` *interpreter*

3.2.4.3 Thread Support for scli

Before the integration work started all `scli` interpreters used to run in a single thread. Communication between the master interpreter and the slave interpreters was accomplished via function calls.

Thread support was added to the `scli` code which allows each interpreter to run in a separate thread. For this purpose the thread API of the `glib`[5] library was used. During `scli` startup `g_thread_init()` is called in order to initialize the thread support of `glib`. Then the master interpreter is created which runs in the main thread. The thread in which an interpreter is running is stored in the `scli_interp` struct which keeps all data associated with the interpreter. Whenever a user invokes the `create scli interp` command the function `create_scli_interp()` is called. This function invokes `g_thread_create()` which creates a new thread and the pointer for this thread is stored in the newly created slave interpreter. The new thread sits in a loop in the

[5] http://developer.gnome.org/doc/API/2.2/glib/

`scli_thread()` function which was added as part of the thread support for `scli`. The main task of `scli_thread()` is to keep polling the command queue for the slave interpreter via calls to `g_async_gueue_pop()`. Whenever a command for a slave interpeter is sent to `scli` it is first handled by the master interpeter. The master interpeter calls `scli_interp_eval()` on this command in order to parse it. Thus, it extracts the name of the slave interpreter that has to execute the command as well as the arguments for the command. The master interpreter creates a message of type `scli_message_t` which consists of the name of the interpreter, the number of arguments (`argc`) and a vector of the arguments (`argv`). The created message is pushed onto the queue of the respective slave interpreter via call to `g_async_queue_push()`. As mentioned above the thread executing the slave interpreter keeps polling this queue and extracts the message via a call to `g_async_gueue_pop()`. The command and the arguments are extracted and the command is executed via a call to `scli_eval_argc_argv()` which is the core function that handles the execution of commands in `scli`. Before the integration work this function used to be called inside the thread in which the master interpreter is running and thus blocking `scli` until the execution of the command is complete. In the multi-threaded version of `scli` the communication between the master interpreter thread and the slave interpreter threads is performed via passing messages of type `scli_message_t`. The communication involved in processing commands by a slave interpreter is shown in Figure 3.2.5. The result is returned via the result field of the `scli_interp` struct.

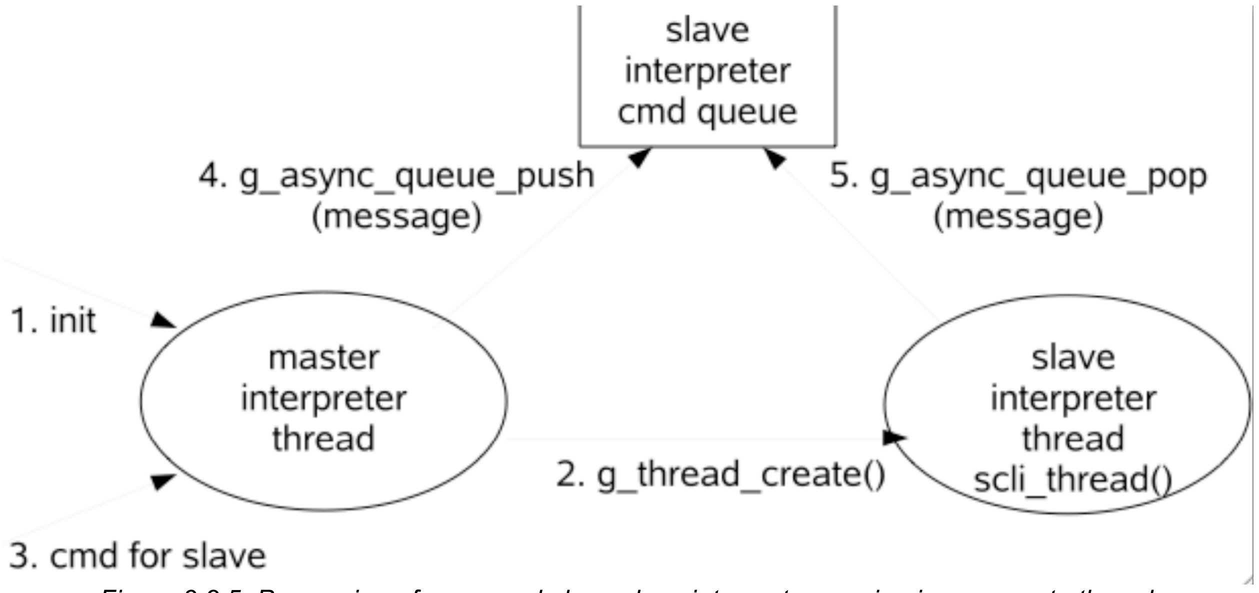

slave interpreter cmd queue
4. g_async_queue_push (message)
5. g_async_queue_pop (message)
1. init
master interpreter thread
slave interpreter thread scli_thread()
2. g_thread_create()
3. cmd for slave

Figure 3.2.5: Processing of commands by a slave interpreter running in a separate thread

When the `scli` program has to be terminated the maste interpreter iterates over all slaves and pushes the command exit onto their queues. This terminates the slave interpreters after which the slave interpreter threads join the master interpreter thread.

In a multithreaded model each interpreter runs in its own thread and the threads are synchronized. All messages will again be initially handled by the master interpreter. An external program using `scli` in order to communicate with a managed device will be able to communicate with several interpreters simultaneously. The multithreaded model will allow several request/response messages to be handled by `scli` simultaneously. Response messages that come from different interpreters can be mixed when sent over the communication channel between the external program and the `scli` master interpreter. Since responses coming from slave interpreters are always prefixed with the interpreter identifier they can be easily matched with request messages when they are received by the external program. Responses coming from the master interpreter will always follow the order of the corresponding request messages because all request messages are processed by the master interpreter serially and the response is returned to the external program before a command is forwarded to the respective slave interpreter (if command is addressed to a slave interpreter).

3.2.4.4 Approaches to Make scli Thread-Safety and Reentrant

Based on the various techniques studied in [24], a top-down approach for making the `scli` code thread safe and reentrant was taken. It is based on the `scli` architecture and consists of the following steps:

- Introduce a global lock in the `scli` code. This will enable multithread support for `scli` but will slow down the performance significantly since only one thread will be able to execute at a time. This solution is similar to the previous status of operation of `scli` and is a fast and simple hack.
- Modify the signatures of reentrant and thread-unsafe functions from the `scli` command implementations. Any static buffers should be eliminated and replaced with buffers provided by the caller. No function should return a pointer to a static buffer. The `scli` command implementations are based on the compiler generated stubs and procedures which are not thread safe. Therefore, after making the `scli` command implementations thread-safe a lock will be introduced at the entry point of the stubs and the MIB procedures. The global lock introduced in step 1 can be removed.
- Modify the `smidump` compiler so that it generates reentrant and thread-safe stubs and procedures. Introduce a lock at the entry point of the `gsnmp` library which is not thread-safe. At this point the lock in the stubs and procedures can be removed.
- Modify the `gsnmp` library so that it is thread-safe and reentrant. Remove all locks. The `gsnmp` library based on the `glib` library which is thread-safe. Therefore, at this point all `scli` code will be thread-safe.

3.2.4.5 Implemented Solutions

Initially we implemented the solution from point 1 of section 3.2.4.4 by introducing a global lock. The function `scli_eval_cmd()` was chosen as an entry point to the `scli` commands implementation. A `G_STATIC_MUTEX` was used as a lock and `g_static_mutex_lock(&scli_global_lock`) was called at the beginning of this function and `g_static_mutex_unlock(&scli_global_lock)` was called just before the function returned. This constitutes a quick and fast hack which however slows down the performance since only one thread can execute at a time.

During the second phase of the implementation the `scli` commands implementation functions were modified in order to make them reentrant. This step involved mainly adapting the formatting interface functions of `scli` many of which either used or returned static char buffers in their implementation. The signatures of those functions were modified so that a reference to a result `GString` buffer was provided by the caller and the return type was changed from `char*` to `gboolean`. The whole formatting API was also changed consistently so that each formatting function contains a reference to a resulting `GString` buffer in its signature. The modification of the formatting API required modifying the code fragments where the formatting functions are called. This code had to be rewritten in order to reflect the new API and deal with allocation and deallocation of return `GString` buffers.

During the next step of the implementation work the stubs and the MIB procedures were made reentrant. For this purpose the `smidump` compiler was modified so that it does not generate static buffers in the implementation of the stubs. This involved modifying several printing functions inside the compiler backend.

The final step of the implementation the global lock was removed from the `scli` command implementations and a lock was introduced at the entry of the `gsnmp` library. Thus, we reached at step 3 from the top-down approach specified in section 3.2.4.4.

In addition to making the functions in the formatting API reentrant and thread-safe a XML formatting API was implemented. These functions mainly deal with formatting data and inserting nodes in the XML tree when XML output is requested. The newly developed API utilizes many of the functions of the formatting API and thus makes handling of XML data easier. Since the formatting API was used during the implementation the XML API is reentrant and thread safe.

3.2.5 Cfengine Accomodation

Cfengine has been modified to provide the necessary hooks for integrating with scli. Scli commands can be entered into cfengine in a specific action section called "scli" using the standard cfengine structural form:

```
scli:
  class_context::
    "scli shell commands"
```

Following IUBs implementation of a protocol in scli for tool intercommunication, a basic implementation of this protocol has been added to the cfengine code base[6] and a syntax based on the existing cfengine interface for shellcommands has been adopted so that scli is treated simply as "another shell". This allows users to pass scli commands with context sensitive parameters and collate the output from within a cfengine framework.

The intermingling of stateful and stateless processes in the dialogue between cfengine and scli makes their integration non-trivial. Several compromises have to be made to aid usability and these could lead to difficulties in the practical use of the tools later. Only experience will be able to determine this.

The basic approach is to open a pipe for the reading and writing of dialogue with the interpreter. There are some limitations to Unix pipes however, in particular input and out-

[6] http://www.cfengine.org

put are separated which makes per-transaction error control essentially impossible. Some work-arounds for this can be explored once the implications are understood by practical deployment.

The use of scli for collecting data for monitoring and machine-learning within cfengine is also, in principle, enabled by the interface using the scli options for producing structured parsable output. However, there remains a significant challenge of testing and using this bridge between the tools. So far, no device data have been identified in snmp implementations that offer sufficient variability and measurement integrity to be useful in this capacity. The effort has also been hampered by HIO's move into a new building which has meant that necessary test equipment has been inaccessible and time has been short. We continue to examine the possibilities inherent in the interoperability of the tools and expect to develop the user interface with the experience of the coming year. The details of the integration will take extended experience to iron out all problems.

3.3 Ponder extensions, packaging and public availability

3.3.1 Software description

Ponder2 is a policy service aimed at the implementation of autonomous self-managed cells (SMC). A SMC is an autonomous collection of software and hardware components such as those in a body-area network, a room or even a large-scale distributed application. Ponder2 comprises a policy interpreter for implementing obligation policies (in the form of event-condition-action rules) a domain-based service where managed objects can be explicitly grouped in hierarchical and overlapping domains and an internal event propagation mechanism for triggering obligation policies. Conceptually the software is derived from past experience with the development of the Ponder system however it has been entirely re-designed to a self-contained implementation that can scale from small embedded systems to larger distributed systems and networks. Because it is self-contained the implementation can be easily embedded into other services.

We have had considerable experience with the use of policies as a means of specifying adaptive behaviour in network management and other applications. The use of interpreted policies means they can be easily changed without shutting down or recoding components. The policy service maintains adapter objects for each of the components on which management actions can be performed. This includes the sensors and other devices present within a SMC, services within those devices and remote SMCs. These adapter objects (also called managed objects) are grouped in a domain structure that implements a hierarchical namespace e.g., similar to a file system. However, unlike in a file system, domains may overlap and a managed object may belong to several domains. Domains and policies are managed objects in their own right on which actions can be performed e.g., adding/removing an object from a domain, enabling or disabling a policy. Consequently, events can trigger obligation policies (ECA rules) that can enable or disable other policies and change domains and domain membership. In essence domains are a means of classifying and grouping the managed objects in a hierarchy and permit them to be addressed using simple path expressions.

When an obligation policy is created in the policy service, an event subscription for that event is sent to an internal event bus. Upon receiving a notification, the policy service evaluates all the obligation policies triggered by that event. Event adapters can also be created for receiving events from external event buses such as Elvin, Siena or XMLblaster.

The policy service has been implemented with particular focus on flexibility and the ability to load all the code needed on-demand. This enables us to use it across a wide variety of applications and devices with different capabilities by only loading those components which are necessary in each case. When started, the policy service has a reference to its root domain and only recognizes the import command that can load new classes. Typically, the classes loaded are factories that permit the creation of new managed objects in domains and the first class to be loaded is the factory for the domain objects themselves (see Figure 3.9a below). This enables the policy service to create new domain objects to form a hierarchy of domains under the root domain. Additional, factory objects are then loaded in order to communicate with external event buses, create policies and create adapters for the various sensors and devices in the SMC. The event factory is specific to the event bus used and encapsulates the protocols necessary to communicate with it. However, multiple event factory objects can be created, allowing the policy service to connect to different event buses with different underlying protocols e.g. XMLBlaster, Siena. Similarly, new types of policies e.g., delegation, filtering, etc. can be defined by providing and dynamically loading the corresponding factory. Adapters can also be created for interacting with specific sensors in a pervasive computing environment. For example a *bsn* factory object encapsulates the code for interacting using IEEE 802.15.4 radio with BSN sensor nodes[7] that are used for eHealth monitoring. Specific factories can then be defined for each of the different types of BSN sensors in use eg. hr sensor for heart-rate monitoring which uses the basic bsn adapter for interaction with BSN nodes.

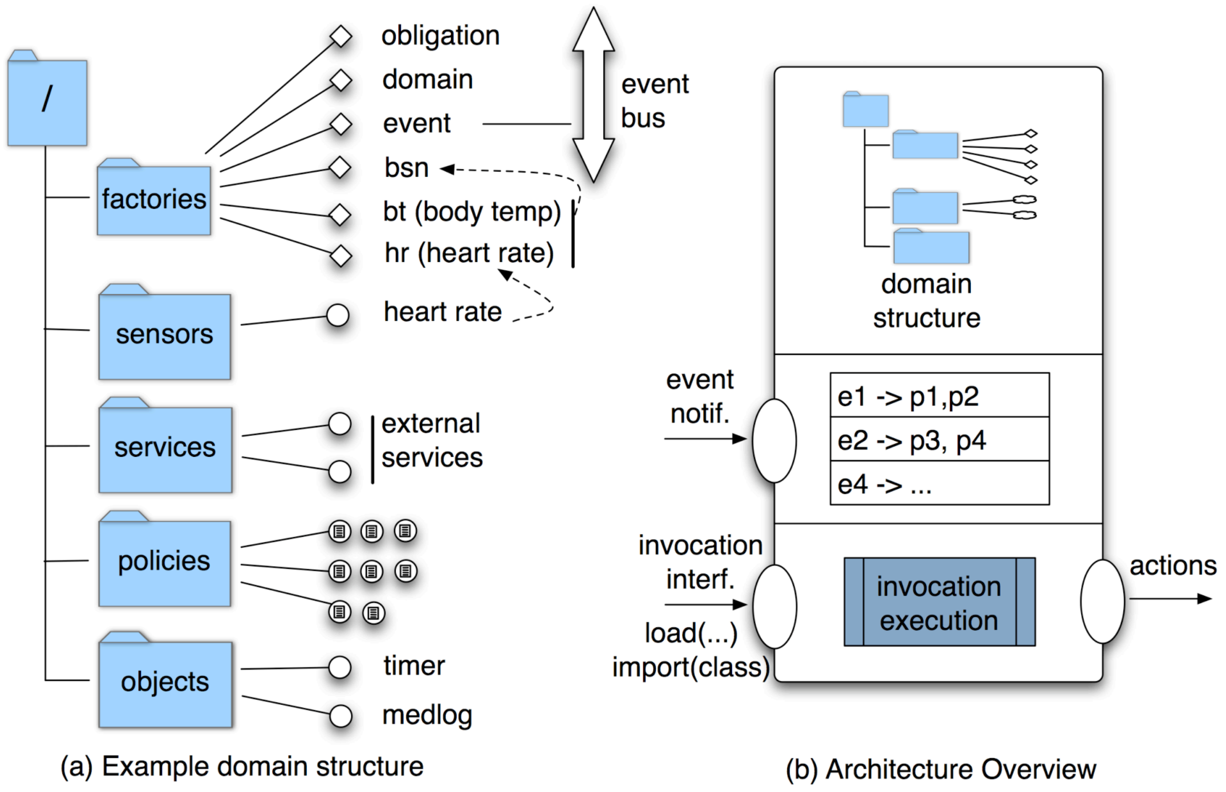


Figure 3.3.1 Policy Interpreter architecture and domain organisation

[7] These BSNs were developed in the DTI UbiMon project (see http://www.ubimon.org). They have very low power 16-bit processors, 64 KB RAM, 256 KB Flash memory, 6 analog channels for sensors and communicate using IEEE 802.15.4 radio

As shown in Figure 3.3.1 the overall architecture of the policy service comprises the domain structure, the table matching obligation policies to events and the execution invocation engine which is used to make the calls to the objects inside the domain structure. Conceptually the policy service has an event interface through which event notifications are received, an invocation interface through which external invocations are received and an action interface through which calls are made to external objects.

XML Programmability

Although we are planning to provide a higher-level declarative language for the specification of policies, Ponder2 currently uses XML in its internal representation as well as its input. Ponder2 interprets XML as a sequence of statements that identify managed object and sub-elements of that XML (typically commands with arguments) that are to be sent to those managed objects. For example the following snippet identifies the root domain ("/") and sends it an add command. The add command has its own structure and information saying what is to be added. From the snippet we can see that the managed object to be added to the root domain will be called *newobject*.

```
<use name="/">
 <add name="newobject">
  ...
 </add>
</use>
```

Event types are defined by executing a command on the appropriate event factory and policy instances are similarly created by using the policy factory as shown below.

```
<use name="/Event">
  <add name="hrGT140">
   <use name="/Template/event">
    <create>
     <arg name="hrvalue"/>
     <arg name="time"/>
    </create>
   </use>
</use>
```

```
<use name="/Policy">
  <add name="obliggt65">
   <use name="/Template/policy">
    <create type="obligation" event="/Event/hrGT140"
active="true">
            <arg name="hrvalue"/>
            <arg name="time"/>
            <action>
            <use name="/warning_alarm" alarm="on"/>
            </action>
    </create>
   </use>
  </add>
 </use>
```

Communications Library

Ponder2 uses a communications library that can support multiple protocols for remote communciations. These protocols are loaded dynamically as required. The interpreter knows nothing about the protocols until it comes across a URL containing an unknown protocol when a remote object is specified. At that time Ponder2 searches to see if an appropriate protocol module exists, if so it is loaded and used. New protocols are relatively simple to write and can be added to an interpreter by including the jar file containing the implementation of the protocol in the interpreter's classpath. Currently provided protocols include RMI and SOAP/Web-service communications.

Interacting with the interpreter

The policy service can be invoked through an RMI interface, a web-service interface or through a command line interface that resembles the Bourne shell. The latter provides commands for navigation of the domains structure in a similar way to the navigation of a file system in UNIX. Additionally it gives the ability to invoke commands upon the managed objects present in the domain hierarchy and accepts XML directives as input, which are then sent directly to the interpreter.

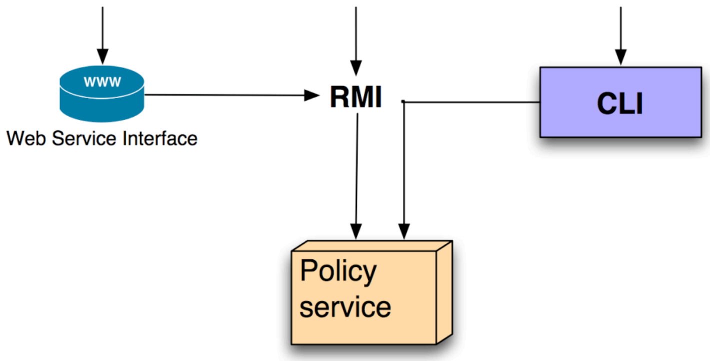


Figure 3.3.2 Interacting with the Policy Service

3.3.2 Detailed list of extensions planed under the support of EMANICS

- Packaging and deployment for open source release under LGPL
- Higher-level policy-language specification
- Adaptations required for integration with software from other EMANICS partners
- Support and maintenance of the Ponder2 system.
- Integration with access control and authorisation policy framework being currently developed in other projects.

3.3.3 Expected impact

Conceptually, the software is based upon the previous Ponder framework which has had several thousand downloads and for which we continue to receive requests more than one year after it has been withdrawn from distribution.

The software will be integrated in the frameworks resulting from 2 EU projects: TrustCoM (security and contract management in Virtual Organisations) and Diadem Firewall on (security management, distributed firewall management and intrusion response) as well as project funded by the UK-EPSRC. A number of organisations from both industry and academia have already expressed an interest in using this software.

In addition to the traditional network management applications to which the previous version of Ponder was aimed, this new version aims to provide a platform for providing adaptation in pervasive systems and building management applications for pervasive systems. The software is currently being used in order to develop platforms that range from Personal Area Networks for e-Health Monitoring to Autonomous Vehicles and Virtual Organisations.

3.3.4 Progress report

The first version of Ponder2 has been extended and packaged for public release together with a comprehensive set of documentation and programming examples. Ponder2 is available from www.ponder2.net .

3.4 OSIMIS porting

3.4.1 Software description

OSIMIS is an object-oriented management platform based on the OSI model [8] and implemented mainly in C++ [4]. It provides an environment for the development of OSI-based management applications which hides the details of the underlying management service/protocol (CMIS/CMIP [9], [10]) through object-oriented Application Program Interfaces (APIs) and allows designers and implementors to concentrate on the intelligence to be built into management applications rather than the mechanics of management service/protocol access. OSIMIS was designed from the beginning with the intent to support the integration of existing systems with either proprietary management facilities or different management models. As such, it also supports a generic CMIS/P to SNMP application gateway [6] for managing SNMP-capable devices. Different methods for the interaction with real managed resources are supported, encompassing loosely coupled resources as is the case with subordinate agents and management hierarchies.

OSIMIS was originally developed in a number of European research projects, namely RACE NEMESYS and ICM and ESPRIT MIDAS and PROOF. It has been used extensively in both research and commercial environments and has served as the management platform for a number of other research projects (RACE, ESPRIT, ACTS, FP6). It has been used extensively for research and in the 1990's it was being used by many research institutions worldwide, distributed by University College London (UCL). After its main developer Prof. G. Pavlou moved to the University of Surrey in the beginning of 1998, the software stopped being ported to latest version compilers and Unix systems and was eventually withdrawn from the public domain. The intention is to make it again publicly available in the context of EMANICS.

3.4.1.1 OSIMIS component overview

OSIMIS uses the ISODE (ISO Development Environment) [21] as the underlying OSI upper layer protocol stack. The OSIMIS services and architecture are shown in Figure 3.4.1. In the layered part, applications are programs while the rest are building blocks realised as libraries. The lower part shows the generic applications provided; from those the ASN.1 and GDMO tools are essential in providing off-line support for the realisation of new MIBs. The thick line indicates all the APIs an application may use. In practice though most applications use only the Generic Managed System (GMS) and the Remote MIB (RMIB) APIs when acting in agent and manager roles respectively, in addition to the Coordination and high-level ASN.1 support APIs. The latter are used by other components in this layered architecture and are orthogonal to them, as such they are shown aside (Figure 3.4.2). Directory access for address resolution may or may not be used, while the Directory Support Service (DSS) API provides more sophisticated searching and discovery facilities.

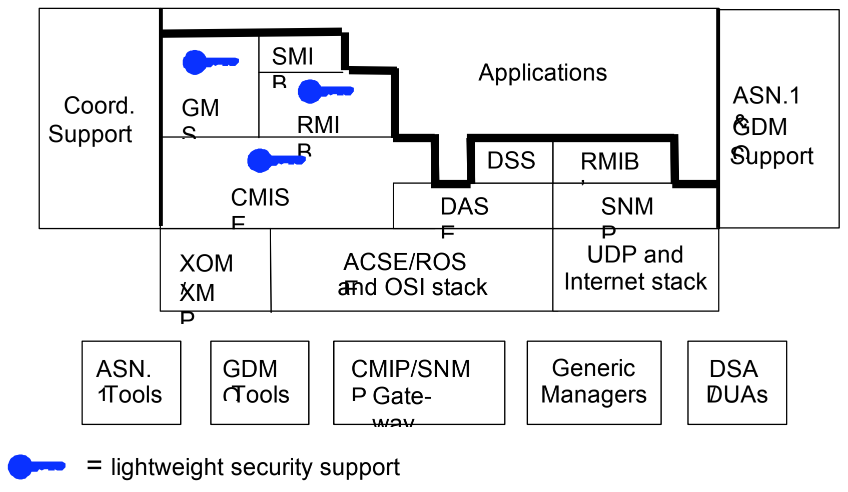


Figure 3.4.1 OSIMIS layered architecture and generic applications

OSIMIS comprises the following types of support:

- high-level object-oriented APIs realised as libraries,
- tools (compilers/translators) as separate programs supporting the above APIs,
- generic applications such as browsers, gateways, directory servers,
- specific useful management applications.

Some of these services are provided by ISODE and these are:

- the OSI Transport (class 0), Session and Presentation protocols
- the Association Control and Remote Operations Service Elements (ACSE and ROSE),
- the Directory Access Service Element (DASE),
- an ASN.1 compiler with C language bindings (the pepsy tool),
- a full Directory Service implementation including an extensible Directory Service Agent (DSA) and a set of Directory User Agents (DUAs).

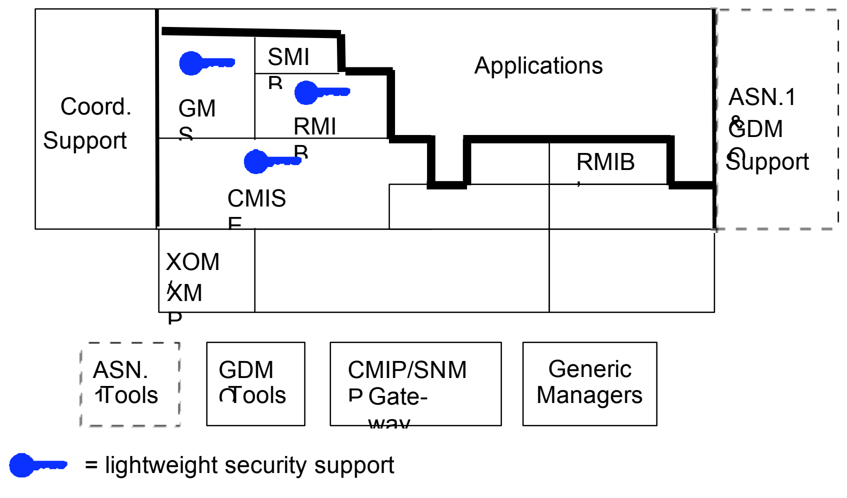


Figure 3.4.2 The components provided by OSIMIS

OSIMIS is built as an environment using ISODE and is mostly implemented in the C++ programming language. The services it offers are:

- an implementation of CMIS/P using the ISODE ACSE, ROSE and ASN.1 tools,

- high-level ASN.1 support that encapsulates ASN.1 syntaxes in C++ objects,
- an ASN.1 Attribute compiler which uses the ISODE pepsy compiler to automate to a large extent the generation of syntax C++ objects,
- a Coordination mechanism that allows an application to be structured in a fully event-driven fashion and can interwork with similar mechanisms,
- the Generic Managed System (GMS) which is an object-oriented OSI agent engine offering a high-level API to implement new managed object classes, a library of generic attributes, notifications and objects and systems management functions,
- a suite of lightweight security mechanisms, which meet many requirements for access control, authentication, data integrity and data confidentiality,
- a compiler for the OSI Guidelines for the Definition of Managed Objects (GDMO) [12] language which complements the GMS by producing C++ stub managed objects covering every syntactic aspect and leaving only behaviour to be implemented,
- the Remote and Shadow MIB high-level object-oriented manager APIs,
- a Directory Support service offering application addressing and location transparency services,
- a generic CMIS/P to SNMP application gateway driven by a translator between SNMP and OSI GDMO MIBs,
- a set of generic manager applications.

3.4.1.2 Underlying ISODE communication environment

The ISO Development Environment (ISODE) [21] is a platform for the development of OSI services and distributed systems. It provides an upper layer OSI stack that conforms fully to the relevant ISO/ITU-T recommendations and includes tools for ASN.1 manipulation and remote operations stub generation. Two fundamental OSI applications are provided: Directory Service (X.500) [7] and File Transfer (FTAM) implementations. ISODE is implemented in the C programming language [5]. The upper layer protocols realised are the transport, session and presentation protocols of the OSI seven-layer model. The Association Control, Remote Operations and Reliable Transfer application layer Service Elements (ACSE, ROSE and RTSE) are also provided which, when used in conjunction with the ASN.1 support, act as building blocks for higher level services.

ASN.1 manipulation is very important to OSI distributed applications. The ISODE approach for a programmatic interface (API) relies on a fundamental abstraction known as *Presentation Element* (PE). This is a generic C structure capable of describing in a recursive manner any ASN.1 data type. An ASN.1 compiler known as *pepsy* is provided with C language bindings, which produces concrete representations i.e. C structures corresponding to the ASN.1 types and also encode/decode routines that convert those to PEs and back. The presentation layer converts between PEs and the encoded data stream according to the encoding rules (e.g. BER).

3.4.1.3 Management protocol and abstract syntax support

OSIMIS is based on the OSI management model as the means for end-to-end management and as such it implements the OSI Common Management Information Service/Protocol (CMIS/P). This is implemented as a C library and uses the ISODE ACSE and ROSE service elements together with their ASN.1 support. Every request and response CMIS primitive is realised through a procedure call. Indications and confirmations are realised through a single ‘wait’ procedure call. Associations are represented as communication end-points (file descriptors) and operating system calls e.g. the Berkeley

UNIX `select`(2) can be used for multiplexing them to realise event-driven policies. The OSIMIS CMIS API is known as the MSAP API, standing for Management Service Access Point. OSIMIS also includes an implementation of the Internet SNMPv1 which is used by the generic application gateway between the two. Applications using CMIS need to manipulate ASN.1 types for the CMIS managed object attribute values, actions, error parameters and notifications.

Regarding ASN.1 manipulation, it is up to an application to encode and decode values as this adds to its dynamic nature by allowing late bindings of types to values and graceful handling of error conditions. From a distributed programming point of view this is unacceptable and OSIMIS provides a mechanism to support high-level object-oriented ASN.1 manipulation, shielding the programmer from details and enabling distributed programming using C++ objects as data types. This is achieved by using polymorphism to encapsulate behaviour in the data types determining how encoding and decoding should be performed through an ASN.1 meta-compiler which produces C++ classes for each type. Encode, decode, parse, print and compare methods are produced together with a get-next-element one for multi-valued types (ASN.1 SET OF or SEQUENCE OF). Finally, the very important ANY DEFINED BY construct is automatically supported through a table driven approach, mapping types to syntaxes. This high-level OO ASN.1 approach is used by higher level OSIMIS APIs.

3.4.1.4 Application coordination support

OSIMIS provides an object-oriented infrastructure in C++ that allows an application to be organised in a fully event-driven fashion under a single-threaded execution paradigm, where every external or internal event is serialised and taken to completion on a ‘first-come-first-served’ basis. This mechanism allows the easy integration of additional external sources of input or timer alarms and it is realised by two C++ classes: the *Coordinator* and the *Knowledge Source* (KS). There should always be one instance of the Coordinator or any derived class in the application, whilst the KS is an abstract class that facilitates usage of the coordinator services and integrates external sources of input and timer alarms. All external events and timer alarms are controlled by the coordinator whose presence is transparent to implementors of specific KSs through the abstract KS interface.

This coordination mechanism is designed in such a way as to allow integration with those provided by other systems; which is achieved through special classes derived from the coordinator, allowing interworking with a particular mechanism. These specialised coordinator classes still control the sources of input and timer alarms of the OSIMIS KSs, but pass on the task of performing the central listening to the other system’s coordination mechanism. This is required for OSIMIS agents which wish to receive association requests, since ISODE imposes its own listening mechanism which hides the Presentation Service Access Point (PSAP) on which new ACSE associations are accepted. A similar mechanism is needed for Graphical User Interface technologies which have their own coordination mechanisms: In which case, a new specialised coordinator class is needed for each of them. The X-Windows Motif, the Tcl/Tk interpreted language and the InterViews graphical object library are fully integrated.

3.4.1.5 The generic managed system

The Generic Managed System (GMS) provides support for building agents that offer the full functionality of the OSI management model, including scoping, filtering, access control, linked replies and cancel-get. OSIMIS fully supports the *Object Management* [14], *Event Reporting* [16] and *Log Control* [17] Systems Management Functions (SMFs); the qualityofServiceAlarm notification of the *Alarm Reporting* [14] SMF; together with profiles of the *Access Control* [20], *Monitor Metric* [18] and *Summarisation* [19] SMFs. In conjunction with the GDMO compiler the GMS offers a very high level API for the integration of new managed object classes where only semantic aspects (behaviour) need to be implemented. It also offers different methods of access to the associated real resources, including proxy mechanisms, based on the Coordination mechanism.

The Generic Managed System is built using the coordination and high-level ASN.1 support infrastructure and most of its facilities are provided by three C++ classes which interact with each other:

- the *CMISAgent*, which provides OSI agent facilities,
- the *MO* which is the abstract class providing generic managed object support,
- the *MOClassInfo* which is a meta-class for a managed object class.

The GMS library also contains generic attribute types such as counter, gauge, counterThreshold, gaugeThreshold and tideMark, and specific attributes and objects as in the Definition of Management Information (DMI) [12], which relate to the SMFs. The object-oriented internal structure of a managed system built using the GMS in terms of interacting object instances is shown in Figure 3.4.3.

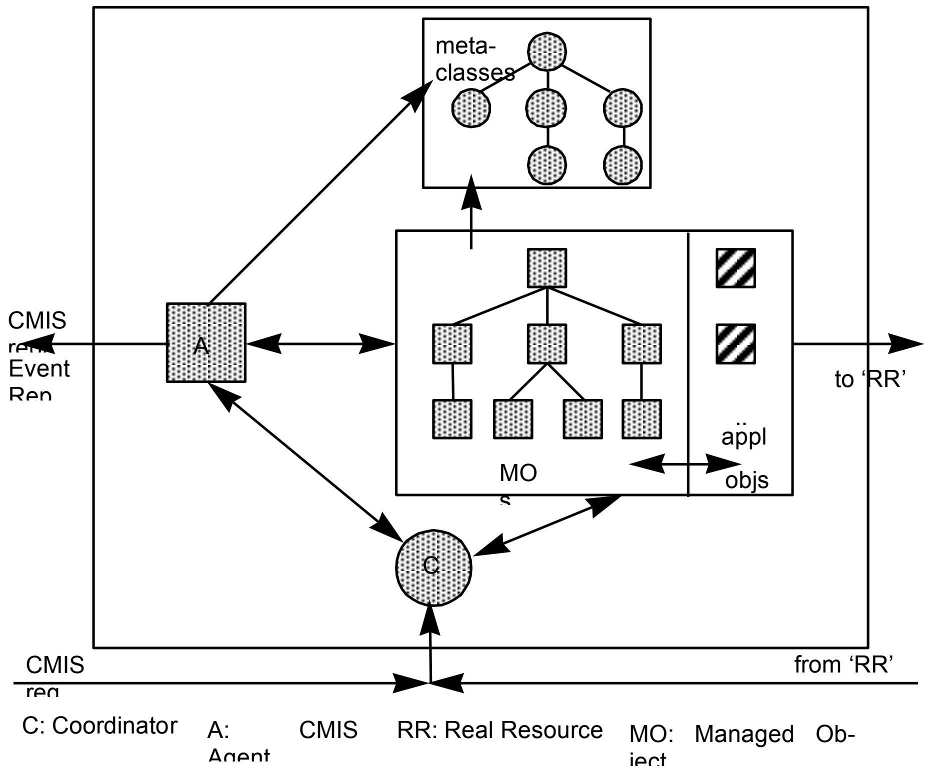


Figure 3.4.3 The GMS object-oriented architecture

Managed object instances and meta-classes

Every specific managed object class needs access to information common to the class which is independent of all instances and common to all of them. This information concerns attributes, actions and notifications for the class, initial and default attribute values, 'template' ASN.1 objects for manipulating action and notification values, integer tags related to the object identifiers etc. This leads to the introduction of a common meta-class for all the managed object classes, the MOClassInfo. The inheritance tree is internally represented by instances of this class linked in a tree fashion as shown in the ‘meta-classes’ block of Figure 3.4.3.

Specific managed object classes are simply realised by equivalent C++ classes produced by the GDMO compiler and augmented manually with behaviour. Through access to meta-class information requests are first checked for correctness and authorisation before the behaviour code that interacts with the real resource is invoked. Behaviour is implemented through a set of polymorphic methods which may be redefined to model the associated real resource. Managed object instances are linked internally in a tree mirroring the containment relationships - see the “MOs” part of Figure 3.4.3. Scoping becomes simply a tree search, whilst special care is taken to make sure the tree reflects the state of the associated resources before scoping, filtering and other access operations. Filtering is provided through compare methods of the attributes which are simply the C++ syntax objects, or derived classes (when behaviour is coded at the attribute level.)

Real resource access

There are three possible types of interaction between the managed object and the associated resource with respect to CMIS Get requests:

- access upon external request,
- ‘cache-ahead’ through periodic polling of the real resource,
- update through asynchronous event reports from the real resource.

The first one means that no real resource access is performed until a managing process accesses the managed object. In the second, requests are responded quickly, especially with respect to loosely coupled resources, but timeliness of information may be slightly affected. Finally the third one is good but only if it can be tailored so that there is no unnecessary overhead when the agent is idle.

The GMS offers support for all methods through the coordination mechanism. When asynchronous reports or results are required, it is likely that a separate object will be needed to demultiplex the incoming information and deliver it to the appropriate managed object instance. It should be noted here that an asynchronous interface to real resources driven by external CMIS requests is not currently supported as this requires an internal asynchronous interface between the agent and the managed objects. These objects are usually referred to as Internal Communications Controllers (ICCs) and are essentially specialised knowledge source objects.

Systems management functions

As already stated, OSIMIS supports the most important of the systems management functions. As far as the GMS is concerned, these functions are realised as special managed objects and generic attribute and notification types which can be simply instantiated or invoked. This is the case, for example, with the alarm reporting, metric and summarisation objects. In other cases, the GMS knows the semantics of these classes and uses them accordingly e.g. in access control, event and log control. Notifications can be emitted through a special method call and all the subsequent notification processing is carried out by the GMS in a fashion transparent to application code. In the case of the object management SMF the code generated by the GDMO compiler together with the GMS completely hiding the emission of object creation and deletion notifications, as well as the attribute change one when something is changed through CMIS. Log control is realised simply through managed object persistency which is a general property of all OSIMIS managed objects. This is implemented using the GNU version of the UNIX DBM database management system and relies on object instance encoding using ASN.1 and the OSI Basic Encoding Rules to serialise the attribute values. Any object can be persistent so that its values are retained between different incarnations of an agent application. At start-up time, an agent looks for any logs or other persistent objects and simply arranges its management information tree accordingly.

Monitoring and summarisation SMFs

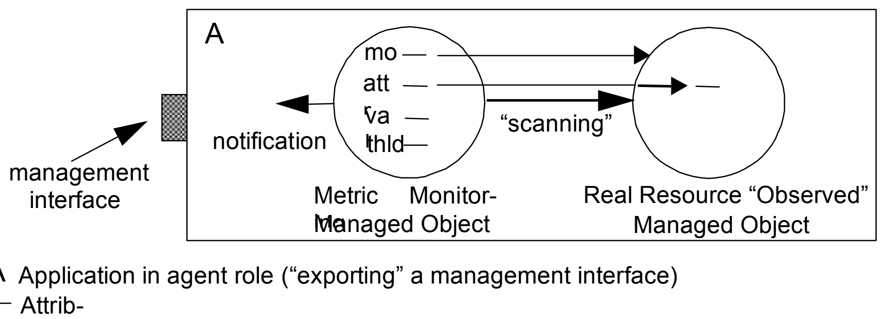
The OSI Structure and Definition of Management Information (SMI/DMI) specify generic attribute types such as counter, gauge, threshold and tide-mark. Gauges model entities with associated semantics e.g. number of calls, users, quality of service etc. or the rate of change of associated counters e.g. bytes per second. Thresholds and tide-marks may be applied to gauges and generate QoS alarms and also attribute value changes, indicating change of the high or low 'water mark'. Such activities are of a *managing* nature. Although thresholding functions could be made part of managed objects modelling real resources, it does not take long to recognise their genericity. As such, they are better provided elsewhere so that they become re-usable. The relevant ISO/ITU-T group recognised the importance of generic monitoring facilities and standardised the Metric Monitor [17] and Summarisation [18] systems management functions. By making such functions generic, it is possible to implement them once and make them part of the associated platform infrastructure.

The whole idea behind monitor metric objects is to provide thresholding facilities in a *generic* fashion. Monitor metric objects may be instantiated within an application in an agent role and be configured to monitor, at periodic intervals, an attribute of another real resource managed object. That attribute may belong to a network element managed object but also to a higher-level management application, being mapped onto lower-level managed objects through an Information Conversion Function (ICF). The observed attribute should be a counter or gauge and the metric object either observes it 'as is' or converts the observed values to a rate (derived gauge) over time. Statistical smoothing of the observed values is also possible if desired.

The main importance of this facility is the attachment of gauge thresholds and tide-marks to the resulting derived gauge which may generate quality of service alarms and indicate the high and/or low 'water mark', as desired by systems using this function. In fact, the metric objects essentially enhance the 'raw' information model of the observed object. The metric monitor functionality can be summarised as:

- *data capture:* through observation or 'scanning' of a managed object attribute,
- *data conversion*: potential conversion of a counter or gauge to a derived gauge,
- *data enhancement:* potential statistical smoothing of the derived result,
- *notification generation:* QoS alarm and attribute value change notifications.

The metric monitoring model is shown in Figure 3.4.4.


A
mo
att
va
thld
notification
"scanning"
management interface
Metric Monitor-Managed Object
Real Resource "Observed" Managed Object

A Application in agent role ("exporting" a management interface)
— Attrib-
ute

Figure 3.4.4 The metric monitoring model

The metric objects offer the OSI management power through event reporting and logging, even if the ‘raw’ observed management information model does not support such notifications. More importantly, they obviate the use of rates, thresholds and tide-marks in a way tied to specific managed object classes but they allow the same flexibility and power dynamically, whenever a managing system needs it. Such a monitoring facility reduces the management traffic between applications and their impact on the managed network by supporting an event-based operation paradigm.

The summarisation objects extend the idea of monitoring a single attribute to monitoring many attributes across a number of selected managed object instances. They offer similar but complementary facilities to metric objects. In this case, there are no comparisons or thresholding but only the potential statistical smoothing and simple algorithmic results of the observed values (min, max, mean and variance). These are reported periodically to the interested managing systems through notifications. The observed managed objects and attributes can be specified either by supplying explicitly their names or through CMIS scoping and filtering. The observed values may be raw ones, modelling an underlying resource, or enhanced values as observed by metric objects. They can be reported either at every observation period or after a number of observation periods (buffered scanning).

3.4.1.6 Generic high-level manager support

The Remote MIB (RMIB)

The Remote MIB (RMIB) support service offers a higher level API which provides the abstraction of a proxy agent object. This handles association establishment and release; hides object identifiers through friendly names; hides ASN.1 manipulation using the high-level ASN.1 support; hides the complexity of CMIS distinguished names and filters through a string-based notation; assembles linked replies; provides a high level interface to event reporting which hides the manipulation of event discriminators and finally provides error handling at different levels. There is also a low level interface for applications that do not want this friendliness and the performance cost it entails but they still need the high-level mechanisms for event reporting and linked replies.

In the RMIB API there are two basic C++ classes involved: the *RMIBAgent* which is essentially the proxy object (a specialised KS in OSIMIS terms) and the *RMIBManager* abstract class which provides call-backs for asynchronous services offered by the RMIBAgent. While event reports are inherently asynchronous, manager to agent requests can be both: synchronous, in an RPC like fashion, or asynchronous. In the latter case linked replies could all be assembled first or passed to the specialised RMIBManager one by one. It should be noted that in the case of the synchronous API the whole application blocks until the results and/or errors are received, or a timeout occurs, whilst this is not the case with the asynchronous API. The introduction of threads or co-routines will obviate the use of the asynchronous API for reasons other than event reporting or a one-by-one delivery mechanism for linked replies.

The Tcl-RMIB API

Being both interpreted and string based, the Tcl language was selected to form the basis for the OSIMIS scripting language *Tcl-RMIB*. The presence of existing string based interfaces to CMIS, via the RMIB, has greatly assisted in the development of this new interface. Tcl-RMIB [10] assists rapid prototyping of management GUIs, event monitors, MIB browsers etc. with similar functionality to that of the RMIB API, supporting both synchronous and asynchronous interactions.

3.4.1.7 The ASN.1 compilers

For each MOC attribute type's syntax, which is specified in ASN.1, the existing ISODE *Pepsy* compiler can produce a C language structure, together with functions for encoding and decoding, printing, string parsing and performing comparisons. High-level C++ abstractions are then used to provide the support for dealing with these ASN.1 based attribute types, in such a way that utilisation of the actual abstract and transfer syntaxes of the ISODE ASN.1 support service are hidden from the implementer by *encapsulating* the Pepsy compiler output. An ASN.1 Attribute compiler can automate the production of the required C++ attribute classes.

3.4.1.8 The GDMO compiler

Being script driven, the OSIMIS GDMO compiler is completely platform-independent i.e. it does not contain any hard-coded knowledge of a specific network management platform, in this case of OSIMIS. Typically a compiler parses a program (of GDMO statements, in this case) and builds up a symbol table representation of the program; in our compiler the symbol table is represented as a set of C++ objects. The code generation phase of the compiler then produces code for the target platform. In our compiler, the code generation phase is controlled by an interpreted script language. The script language is able to address the information in the compiler's symbol table and to output the information to files that conform to the requirements of a particular network management platform; in software engineering terms, this is achieved by adding meta class information to the C++ symbol table objects so that they can be used as variables in the script language. In short, the symbol table data of the compiler is entirely malleable. It is in this sense that the knowledge of a particular network management platform is not hard-coded in the GDMO compiler.

3.4.1.9 Generic managers

There is a class of applications which are semantic-free and these are usually referred to as MIB browsers as they allow one to move around in a management information tree, retrieve and alter attribute values, perform actions and create and delete managed objects. OSIMIS provides a MIB browser with a Graphical User Interface based on the InterViews X-Windows C++ graphical object library. This allows management operations to be applied and also provides a monitoring facility. Its successor, which is to be re-engineered in Tcl/Tk, will also have the capability of receiving event reports and of monitoring objects through event reporting.

In addition OSIMIS provides a set of programs that operate from the command line and realise the full set of CMIS operations. These may be combined together in a 'management shell'. There is also an event sink application that can be used to receive event reports according to specified criteria. Both the MIB browser and these command line programs owe their genericity to the generic CMIS facilities (empty local distinguished name { for the top MIB object, actualClass and scoping) and the manipulation of the ANY DEFINED BY ASN.1 syntax through a table driven approach.

3.4.1.10 Sample agent

OSIMIS contains a non-standard implementation of an OSI Transport Protocol (TP) MIB. This manages the TP implementation of the ISODE stack and has proved to be very useful in monitoring the activity of ISODE based applications, such as DSAs, MTAs, transport bridges and even management applications themselves. It also includes an agent with an example object modelling the Unix operating system (*uxObj1*) as a trivial example managed object.

3.4.1.11 Distributed processing example

To demonstrate the use of the OSI management model as a powerful paradigm for distributed processing when supported by high-level object-oriented APIs, the OSIMIS platform was extended with a distributed processing example. A *simpleStats* class was specified providing simple statistical services in the form of actions. The currently supported services (actions) are:

- *calcSqrt* - returns the square root of a non-negative real number
- *calcMeanStdDev* - calculates and returns the mean and standard deviation of a series of real numbers

The generic *maction* utility can be deployed to provide a 1 line shell script (!) that provides access to these services. In case a more complex client program needs to utilise these services, the RMIB access API may be used; a demonstration client is included which required only 20 lines in C++. The OSIMIS location transparency service may be used to hide the location where an instance of *simpleStats* object executes.

3.4.1.12 The generic CMIS/P to SNMP gateway

The industry standard for network element management is the Internet SNMP, which is less powerful than the OSI CMIP. The same holds for the relevant information models; the OSI is fully object-oriented while the SNMP supports a simple remote debugging paradigm. Generic application gateways between them are possible without any semantic loss for conversion from CMIS/P to SNMP as the latter's operations and information model are a subset of the OSI ones. Work for standards in this area has been driven by the Network Management Forum (NMF) while the ICM project contributed actively to them and also built a generic application gateway.

This work involves a translator between Internet MIBs to equivalent GDMO ones (i.e. the `imibtool`) and a special script for the GDMO compiler which produces run-time support for the generic gateway. That way the handling of any current or future MIBs will be possible without the need to change a single line of code. It should be added that the generic gateway works with SNMPv1 but may be extended to cover SNMPv2.

A fundamental design requirement for the gateway is to achieve seamless interoperability between TMN management Operations Systems (OS) or Mediation Functions (MF) and SNMPv1 managed resources. An efficient mapping is essential given the fact that the gateway introduces an intermediate hop in the manager/agent communication path, see Figure 3.4.5.

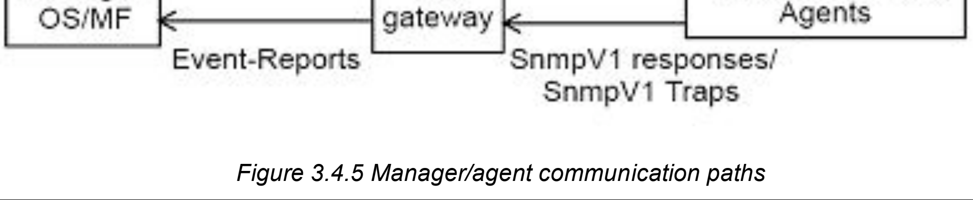


Figure 3.4.5 Manager/agent communication paths

3.4.2 Detailed List of extensions planed under the support of EMANICS

As already stated, this software is not operational anymore since the late 1990's. In addition, it is not available anymore in the public domain. So the key extension is to port it to the latest version C/C++ compilers and to make it publicly available as an example OSI management implementation and platform for building applications. The last publicly available version was running on Solaris and used an early version of the GNU compilers. Earlier versions had also run on SunOS and HPUX. As such, the key targets are the following:

- Port it to the latest version GNU C/C++ compilers under Linux and Solaris.
- Make publicly available both OSIMIS and ISODE under a Lesser General Public Licence (LGPL) through a Web site that will be created.

The key target will be to re-instate the core parts of the infrastructure that include generic agent and manager infrastructure and key agent applications. Additional developments will target the addition of new important infrastructure. The detailed list of tasks is the following.

- Task 1: Port the ISODE environment to the latest compiler versions on Linux and Solaris, this will be made available separately.
- Task 2: Port the OSIMIS core components to the latest compiler versions on Linux and Solaris. These will include the ASN.1/GDMO compilers, the generic agent infrastructure, the example agents (unix MIB, OSI transport protocol MIB, example distributed processing MIB) and the generic command line managers.
- Task 3: Port additional parts of the infrastructure that have not been ported for a long time. These will include the generic CMIS/P to SNMP gateway, the Tcl-CMIS manager API and possibly the Motif/InterViews-based generic MIB browser.
- Task 4: Distribute a beta version to EMANICS partners to be tried and tested.
- Task 5: Make both OSIMIS and ISODE software available through relevant Web pages which will be constructed.
- Task 6: Build a native agent for (parts of) the GDMO version of TCP/IP MIB-II as in RFC1214 as a realistic specific agent for experiments.

3.4.3 Expected impact

OSI-SM is taught in many Network Management courses and the software is expected to be used in many Universities for teaching / demonstration purposes. In addition, given that OSI-SM is still used in telecommunications environments, it is likely that developers will acquire it. Given its previous popularity, it should enhance the EMANICS visibility. Finally, it will be used by EMANICS partners for additional experimentation and comparisons.

It should be finally added that the software was initially developed in the EU ESPRIT INCA, PROOF and MIDAS projects and the RACE NEMESYS and ICM ones. Being a key result of European research, EMANICS provides an excellent opportunity to also expose to the wider community previous European project results.

3.4.4 Progress report

Updates & changes

This subsection has been extended and rewritten to report the extensions and progress made in the last 6 onths of phase 1.

OSIMIS uses the ISODE upper layer and ASN.1 support, so ISODE had to be ported first. Then OSIMIS was ported, using the ISODE upper layer stack library (libisode.a) and the ASN.1 compiler (pepsy). Both environments have been ported to both Linux and Solaris. Porting to Linux was done using Fedora Core 4 and the GNU gcc and g++ compilers version 4.0.2. Porting was also done to a recent Solaris version using the GNU gcc and g++ compilers version 3.6.
ISODE entails an upper layer OSI stack library that operates over different lower layers, including TCP/IP using the RFC1006 ISO Services Over TCP/IP method, an ASN.1 compiler and an X.500 Directory Service known as QUIPU which may be used to provide application name to (presentation) address resolution. All these aspects of ISODE have been ported and the relevant archive provided is isode-8.0.5.tar which provides the original ISODE-8.0 source archive and 5 patch archives. An ISODE-README file provides information on the porting and gives direction for building the relevant libraries and programs, pointing also to the isode-8.0/READ-ME file for additional information. It should be noted that the ISODE is a useful environment in its own right as a development environment for OSI applications and as such it is available separately.
OSIMIS is an object-oriented environment for the development of OSI Management applications and entails generic agent and manager infrastructure, an object-oriented ASN.1 compiler that wraps up the ISODE pepsy compiler, a GDMO compiler that uses the object-oriented ASN.1 compiler, specific agents and a generic CMIS/P to SNMP application-level gateway. All these parts have been ported, so more specifically the ported OSIMIS version entails the following:

- An implementation of the CMIS/P protocol (libmsap.a);
- Object-oriented ASN.1 compiler and GDMO compiler;
- Generic agent support through the Generic Managed System (libgms.a);
- Generic manager support through the Remote MIB (librmib.a);
- Specific agents for infrastructure testing (sma and odpserv);
- Generic command-line managers providing CMIS capabilities (mibdump, mset, maction, mcreate, mdelete, evsink, evlog);
- and Generic CMIS/P to SNMP gateway, the Internet Q Adaptor (iqa).

All these aspects of OSIMIS have been ported and the relevant archive provided is osimis-4.0.12.tar which provides the original OSIMIS-4.0 source archive and 12 patch archives. An OSIMIS-README file provides information on the porting and gives direction for building the relevant libraries and programs, pointing also to the osimis-4.0/README file for additional information.
OSIMIS and ISODE can be downloaded from :
http://www.ee.surrey.ac.uk/Personal/G.Pavlou/projects/Emanics/osimis/
which is also pointed from the EMANICS site http://www.emanics.org/ .

3.5 *Integrating Network Patterns into cfengine*

Updates & changes

This section is entirely new. It reports the integration made by HIO and KTH on Patterns in cfengine.

3.5.1 Magnitude of the project

The idea of integrating network patterns into cfengine seemed at the outset to be a relatvely straightforward idea, if one could understand the ways of forming logical overlayes at the peer to peer level for hosts. However, in preparing for this work we adopted a systematic approach of investigating approaches already available in cfengine to see if actual extension of the code base was necessary. It turns outt hat several key parts of network patterns can be built from the existing experimental infrastructure of cfengine, and the open source development needed for the initial phase was smaller than expected. The time it took to perfect the experimental infrastructure and test it was much greater than anticipated.

3.5.2 Development work

A rational approach to implementation seemed to be to begin with a model. Promise theory is the modelling tool that was created to describe cfengine's approach, so we have used this. The full details of this work will result in a master's thesis by Matt Disnet at Oslo University College, based on the KTH+HIO collaboration.

By modifying cfengine to more explicitly display promise theory constructs (these already existed but were originally written without the achor of a model) we could use the modelling framwork to describe patterns and then implement them in cfengine code more directly.

Having understood the direct translation of promise concepts into cfengine code, we discussed different ways of building spanning trees for data aggregation using the pull based, promise-based architecture. A number of inadequacies in the experimental features had to be improved and the appropriate functions for data aggregation using simple syntactic representations of the network patterns had to be made so that the structures could be programmed into the existing language, using the idea of context sensitivity.

The unexpected abilty to use context sensitivity in the peer to peer policy rules allowed us to tie this work into work package 9, autonomic computing deliverables also.

In the limited time available for working on the EMANICS deliverables, we have successing in reproducing certain aspects of the network patterns without the adaptive robustness features. We have been methodical rather than hurried in our work, hence we have performed extensive experiemental tests on the properties of the cfengine implementation of patterns. These will results in several papers over the next six months.

The parts missing from the original intention to integrate patterns and cfengine relate to the GAP automated tree creation. Some partial approximations for this are implemented as GetPeerNeighbours, GetPeerLeader functions in cfengine, with failover options. However, the full details of the algorithm still have to be understood. This will probably take another six months to a year to find the time to complete.

3.5.3 Results of the investigations

Patterns are dissemination and aggregation algorithms that bridge theworlds of centralized monitoring and fully distributed monitoring. The extreme cases are chain and star networks. Trees are structures that bridge these two extremes. We can characterize patterns by their depth and breadth. A chain has maximum depth (only one node at each distance from the source) and a star has maximum bredth (all nodes are only one hop from the source).
Here we consider only two patterns: Echo and Gap and consider how these can be implemented in cfengine using existing context awareness within the system.

3.5.3.1 Echo

In the echo pattern, a signal initiated from a single source spreads epidemically through a bounded network using a diverging spanning tree to relay a requests. When the signal reaches the edge of the network, actual responses are passed back along the tree to each parent node, where they are aggregated before being passed up to the next level. The echo pattern therefore forms a wave, spreading out from an epicentre to the edge of the network and back, collecting data as it goes. Echo is intrinsically a "push" protocol.

3.5.3.2 Gap

In the Generic Aggregation Protocol, we assume the existence of a spanning tree. Data are then passed from the leaves up towards a central point which acts as a sink for the data. In some ways, this behaviour is like the return phase of the echo pattern.

3.5.3.3 The topology manager

A key feature of the patterns above is the algorithm by which the topology of the spanning tree is decided. The GAP algorithm, as described by Stadler et al incorporates the topology adjustment mechanism into the GAP aggregation algorithm, hence combining these features into a robust protocol. However, they can be separated also. The GoCast algorithm finds such a spanning tree, for example. At this stage of the work we shall not attempt to encode automated topology management, as this requires additional subsystems. Rather we consider how patterns can be used at the logical level for distributed load balancing, using existing mechanisms within cfengine.

3.5.4 Cfengine and patterns

Patterns are motivated by the need for scalable dissemination and aggregation of data in a network. They were designed primarily with routing nodes in mind. However, the autonomic computing scenario is a tantalizing application for this kind of algorithm.

Cfengine is an open source software system for autonomic host management. A cfengine host is by default a completely autonomous entity with no obligations towards other agents in a physical network. Every node is therefore individual and is not part of a pattern a priori. To use patterns as a form of inter-peer collaboration, we must therefore encode them as policy rules. So, instead of thinking of patterns as being overlays to a physical network one can, for instance, think of them as algorithms for balancing (spreading) processing across an array of agents in a logical overlay network.

There are several questions to be answered about this:

1) Is the underlying physical network topology important in building a logical load sharing topology?

2) How will the topology be decided?

3) How will the topology respond to the failure of nodes?

We are currently undertaking detailed experimental studies to determine the answers to these questions. These studies are expected to be completed in July.

3.5.4.1 Promise agreements and voluntary cooperation

Promise theory is a way of modelling networks of agents cooperating in an ad hoc fashion. Cfengine can be viewed as a reference implementation of the abstract promise-theoretic scenario. (Promise theory was introduced precisely as a modelling framwork that could describe cfengine, where others could not.)
The key features of promise theory are :

1) There exists a collection of indepedendent agents.

2) Agents are autonomous. They make their own decisions and no agent can be forced to perform any function by another. (Actions and decisions made by an agent are entirely voluntary.) i.e. they collaborate only as a matter of individual policy.

3) Each agent has private knowledge. It can share its knowledge with another agent by promising to reveal it (in the manner of a service).

4) There are two kinds of promises called + and -, or "service" and "use" promises. A + promise is a promise to give or provide some kind of abstract service. A - service is a promise to make use of an abstract service (if available)

Promise theory allows us to see the relationship between netw rk patterns and policy for autonomous agents. Each arrow in the promise graph attaches to a rule in the policy to either grant access to data or to fetch available data. In this way we can build dissemination prosesses over graphs using node location data or context sensitivity information.

A common mistake is to think of promises as communication transactions, rather than as abstract behavioural specifiers. A promise says nothing about the details of what is communicated between agents.

We write a promise as a directed graph from one agent to another.

$$\begin{tikzcd} A\_1 \arrow[r, "b"] & A\_2 \end{tikzcd}$$

b is called the promise "body" and contains details of what is being promised. An agent can only promise something about its own behaviour, not about a third party.

A reliable binding between two hosts requires both a promise to serve and a promise to use the service.

$$\begin{tikzcd} A\_1 \arrow[r, "+b"] & A\_2 \end{tikzcd}$$
$$\begin{tikzcd} A\_1 & A\_2 \arrow[l, "-b"'] \end{tikzcd}$$

3.5.4.2 Using context awareness for/form patterns

A method in cfengine is like a pair of promises, provided it is voluntarily declared by both parties. MD5 hash is used to verify that the methods are in fact the same.

The first (service) promise identifies the function being performed, as the body b(). The class expression A_1:: says that this rule applies to the context of agent A_1, which is the service provider (server host). The server=A_1 attribute matches the context expression and, from this, the agent deduces that it is the provider.

```
methods:

  A_1::

     b(params) server=A_1

             +b
   A_1 -------------> A_2
             -b
   A_1 <------------  A_2
```

The second part applied to agent A_2 and has the form:

```
methods:

  A_2::

     b(params) server=A_1

             +b
   A_1 -------------> A_2
             -b
   A_1 <------------  A_2
```

This identifies the function being performed and signals to A_2 that it will use the results performed by server A_1. Since this is not its own identity, this implies that the result is a use-promise.

If we assume that two agents use an identical configuration specification, then a remote procedure call binding can then be written :

```
methods:

  A_1|A_2::

     b(params) server=A_1
```

The same text either in both contexts and a single link in a logical overlay network is added.

3.5.4.3 Promise Chains (forwarding)

```
##########################################################
#
# CHAIN 6 machines 1,2,3,4,5,6 (promise chain)
#
##########################################################

classes:

 always = ( any )

 leaf   = ( node6 )
 root   = ( node1 )

##########################################################

control:

  workfile   = ( "/tmp/chain-pattern" )

##########################################################

methods:

  #
  # Pattern has to be coded in classes (from) and servers (to)
  #

node1|node2::              # U(p) | p - binding

  Aggregate("$(workfile)")

     server=node2
     action=method_pattern.cf
     returnvars=ret
     returnclasses=chain_link

node2|node3::

  Aggregate("$(workfile)")

     server=node3
     action=method_pattern.cf
     returnvars=ret
     returnclasses=chain_link

node3|node4::
  Aggregate("$(workfile)")

     server=node4
     action=method_pattern.cf
     returnvars=ret
     returnclasses=chain_link

node4|node5::

  Aggregate("$(workfile)")

     server=node5
     action=method_pattern.cf
     returnvars=ret
     returnclasses=chain_link

node5|node6::

  Aggregate("$(workfile)")

     server=node6
     action=method_pattern.cf
     returnvars=ret
     returnclasses=chain_link


##########################################################

editfiles:

   !leaf::

   { $(workfile)

   AutoCreate
   EmptyEntireFilePlease
   AppendIfNoSuchLine "$(Aggregate.ret)"
   # Handle errors so no strange loops
   ReplaceAll "Aggregate.ret" With "FAILED"


  leaf::

   { $(workfile)

   AutoCreate
   EmptyEntireFilePlease
   AppendIfNoSuchLine "$(value_loadavg)"


##########################################################

alerts:

  root.Aggregate_chain_link::

   "Chain aggregate $(n)$(host)=$(value_loadavg) at $(date) $(Aggregate.ret) "
```

Simplification for management ease :

```
##########################################################
#
# Netlab config
```

```
#
##########################################################

classes:

 leaf   = ( netlab4 )
 root   = ( netlab1 )

##########################################################

control:

  workfile   = ( "/tmp/chain-pattern" )
  tempfile   = ( "/tmp/chain-temp" )

netlab1::

  serve = ( netlab3 )

netlab3::

  serve = ( netlab4 )


##########################################################

tidy:

##########################################################

copy:

!leaf::

  $(workfile)

    dest=$(tempfile)
    server=$(serve)
    type=checksum

    define=success
    elsedefine=failure

##########################################################

editfiles:

  success::

   { $(workfile)

   AutoCreate
   EmptyEntireFilePlease
   InsertFile "$(tempfile)"
   AppendIfNoSuchLine "copy-chain $(host)=$(value_loadavg) at $(date)"


  failure::

   { $(workfile)

   AutoCreate
   EmptyEntireFilePlease
   AppendIfNoSuchLine "copy-chain - no response from $(serve)"
   AppendIfNoSuchLine "copy-chain $(host)=$(value_loadavg) at $(date)"


  leaf::
   { $(workfile)

   AutoCreate
   EmptyEntireFilePlease
   AppendIfNoSuchLine "copy-chain $(host)=$(value_loadavg) at $(date)"


#############################################################

alerts:

  success::

   "Chain update succeeded"
   PrintFile("$(workfile)","6")

  failure::

 "No Chain update at $(date)"
```

3.5.4.4 Promise Trees (aggregation)

The first example of a 2 into 1 aggregation is given below.

```
##########################################################
#
# Netlab config - depth aggregation (promise tree)
#
##########################################################

classes:

 leaf         = ( netlab3 netlab4 )
 aggregator   = ( netlab1 )

##########################################################

control:

  workfile   = ( "/tmp/chain-pattern" )


  children = (
             A(netlab1,"netlab3,netlab4")
             A(netlab3,"netlab3,netlab4") # This completes contract
             A(netlab4,"netlab3,netlab4")
             )

##########################################################

methods:

  #
  # Pattern has to be coded in classes (from) and servers (to)
  #

netlab1|netlab3|netlab4::               # U(p) | p - binding

  Aggregate("$(workfile)")

     server=$(children[$(host)])
     action=method_pattern.cf
     returnvars=ret
     returnclasses=chain_link

##########################################################

editfiles:

   aggregator::

   { $(workfile)

   AutoCreate
   EmptyEntireFilePlease
   AppendIfNoSuchLine "$(Aggregate_1.ret)"
   AppendIfNoSuchLine "$(Aggregate_2.ret)"
   # Handle errors so no strange loops
   ReplaceAll "Aggregate.*ret" With "FAILED"


  leaf::

   { $(workfile)

   AutoCreate
   EmptyEntireFilePlease
   AppendIfNoSuchLine "$(average_loadavg)"


##########################################################

alerts:

  aggregator.(Aggregate_1_chain_link|Aggregate_2_chain_link)::

   "Chain aggregate $(n)$(host)=$(average_loadavg) at $(date) $(Aggregate_1.ret) $(Ag-
gregate_2.ret) "
```

An approximation to this with greater trust allows us to reduce the management overhead somewhat.

```
##########################################################
#
# Netlab config - breadth aggregation by trusting pull
#
##########################################################

classes:

 leaf   = ( netlab4 netlab3 )
 root   = ( netlab1 )

##########################################################

control:

  Split      = ( , )
  workfile   = ( "/tmp/chain-pattern" )
  tempfile   = ( "/tmp/chain-temp" )

 #                                1
 # One link in a binary tree     / \   aggregation
 #                              3   4

netlab1::
  serve = ( "netlab3,netlab4" )

##########################################################

copy:

!leaf::

  $(workfile)

    dest=$(tempfile)_$(this)
    server=$(serve)
    type=checksum

    define=success
    elsedefine=failure

##########################################################

editfiles:

  success::

   { $(workfile)

   AutoCreate
   EmptyEntireFilePlease
   InsertFile "$(tempfile)_$(serve)"
   AppendIfNoSuchLine "copy-chain $(host)=$(value_loadavg) at $(date)"


  failure::

   { $(workfile)

   AutoCreate
   EmptyEntireFilePlease
   AppendIfNoSuchLine "copy-chain - no response from $(serve)"
   AppendIfNoSuchLine "copy-chain $(host)=$(value_loadavg) at $(date)"


  leaf::

   { $(workfile)

   AutoCreate
   EmptyEntireFilePlease
   AppendIfNoSuchLine "copy-chain $(host)=$(value_loadavg) at $(date)"


#############################################################

alerts:

  success::

   "Chain update succeeded"
   PrintFile("$(workfile)","6")

  failure::

   "No Chain update at $(date)"
```

3.5.4.5 The Echo Pattern

Cfengine's main modus operandi is to "pull" data rather than to push. This is a natural side effect of its philosophy of voluntary cooperation. Push is disallowed. There is one exception however. We are allowed to send a single invitation to each peer to execute its existing policy using the command cfrun. The host is free to disregard this message, but for cooperation purposes it is normal for the peer to respond to such an invitation by executing its policy compliance-checking agent. We can use this mechanism to start an echo avalanche, with a pre-arranged pattern.

The start host executes cfrun to a number of "children". Each child then voluntarily executes cfengine, which in turn encapusulates the execution of cfrun directed at another set of children, which encapsulates cfrun to another set, and so on. Since cfengine aggregates the data from encapsulated processes automatically, it automatically aggregates the entire tree in a synchronized manner. This is the simplest implementation of echo which uses context sensitivity to identify parent-child relationships.

The use-promises are encoded as follows:

```
control:
     actionsequence  = ( shellcommands tidy )
     domain          = ( cftestnet )
     IfElapsed = ( 1 )
     TrustKeysFrom   = ( 10.0.0 )

     node1::
          serve = ( node2:node3:node4 )

     node2::
          serve = ( node5:node6:node7 )

     node3::
          serve = ( node8:node9:node10 )

     node4::
          serve = ( node11:node12:node13 )

     node5::
          serve = ( node14:node15:node16 )

     node8::
          serve = ( node17:node18:node19 )

     node11::
          serve = ( node20 )


classes:
     HasChildren = ( IsDefined(serve) )

shellcommands:
     any::
          "/bin/echo $(hostname)"

     HasChildren::
          "/usr/local/sbin/cfrun $(serve) 2>&1 > /tmp/echorun.$$" background=true
          "/usr/bin/pgrep cfrun > /dev/null; while [ $? = 0 ]; do pgrep cfrun >
/dev/null; done"
          "/bin/cat /tmp/echorun.*"

tidy:
     HasChildren::
          /tmp pattern=echorun.* age=0
```

A sample output execution is given in appendix C.

3.5.5 Future Work

Tests are proceeding as to the most appropriate way for deciding a topology in a cfengine peer network. The criteria for this are somewhat different to the original motivation for patterns. Once these tests have resulted in a basic understanding of the problem, we expect to add topology automation to the tool.

The syntax of cfengine's voluntary cooperation model is based on peer to peer interactions, just like promise theory. This results in clumsy and cumbersome policy files for encoding patterns. Further work is expected to be able to enable regular expressions of some form to more efficiently encode the bilateral promises required for pattern policies.

3.6 NDPMon : An Open Source IPv6 Neighbor Discovrey Monitoring Protocol

Updates & changes

This section is entirely new.

NDPMon is an equivalent of ArpWatch for IPv6 and was developped within the MADYNES Project, a research team from the LORIA - INRIA Lorraine in France. NDPMon, Neighbor Discovery Protocol Monitor, is a tool working with ICMPv6 packets. NDPMon observes the local network to see if nodes using neighbor discovery messages behave properly. When it detects a suspicious Neighbor Discovery message, it notifies the administrator by writing in the syslog and in some cases by sending an email report. Initially developed in a cooperation between INRIA and CISCO Corporation, the packaging and distribution of the software was supported by EMANICS.

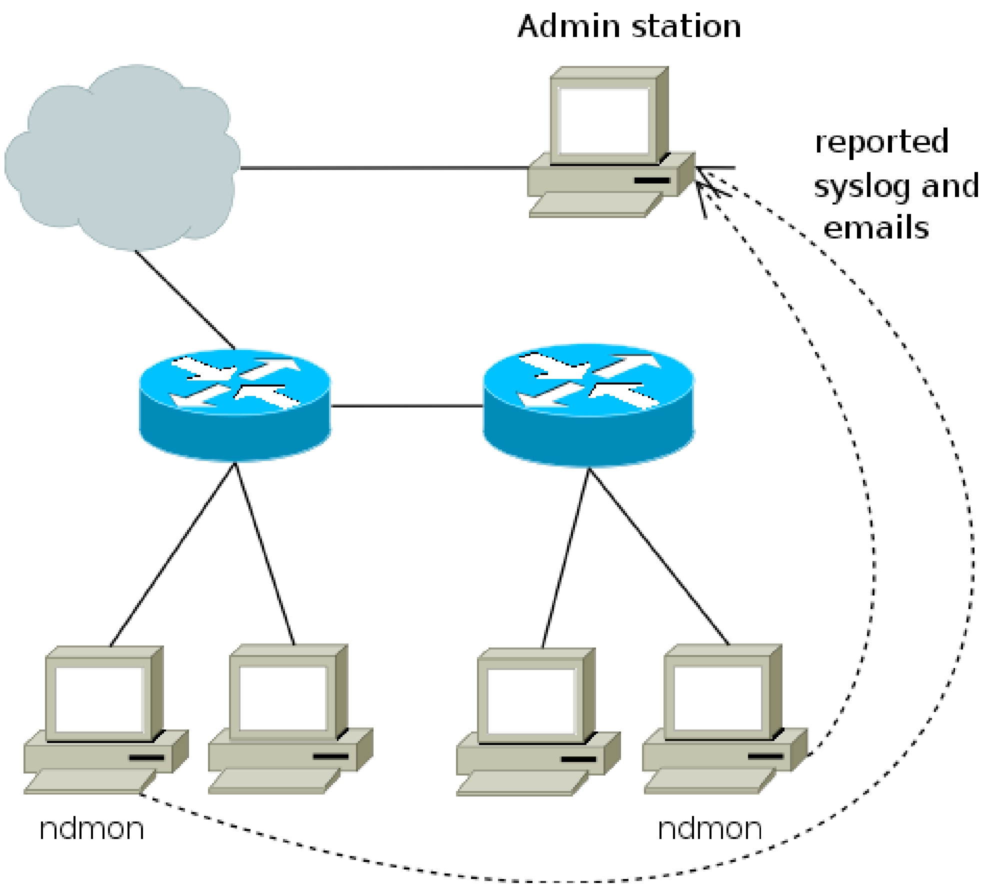


Figure 3.16: NDPMon nodes interaction

NDPMon is very similar to ArpWatch concerning reported activities and erroneous configurations, but it also provides new features, specific to the Neighbor Discovery protocol, for which it detects attacks, which could harm the network. Different kinds of activities can be detected:

The following activities are reported by the system :

- wrong couple MAC/IP
- wrong router MAC
- wrong router IP
- wrong prefix
- wrong router redirect
- router flag in Neighbor Advertisment: NDPMon is carefull about nodes sending router advertisments - only nodes specified to be official routers in the configuration file can send one.
- Duplicate Address Detection DOS
- flip flop
- reused old ethernet address: other kinds of malicious behaviours.

The following activities are Sysloged by the system :

- Unknown MAC MAnufacturer
- new station

- new IPv6 Global Address
- new Link Local Address
- wrong couple MAC/IP
- wrong router MAC
- wrong router IP
- wrong prefix
- wrong router redirect
- router flag in Neighbor Advertisment: NDPMon is carefull about nodes sending router advertisments - only nodes specified to be official routers in the configuration file can send one.
- Duplicate Address Detection DOS
- flip flop
- reused old ethernet address: other kinds of malicious behaviors
- Ethernet mismatch
- IP Multicast
- Ethernet Broadcast.

NDPMon can also be launch with an option disabling reports. This learning phase allows to build the neighbor database during the first execution without raising unappropriate warnings.

The NDPMon software is implemented in C language. It uses libpcap to get and filter neighbor discovery packets and does after different tests. Two XML files are used :

- The first file contains configuration settings like official routers settings or the email address of the admin.
- The second file (that behaves like a cache) contains the list of all neighbors seen by NDPMon on the local network. This cache keeps the IP address, MAC address, and the last time of activity for each node. This list is updated automaticaly during the execution and saved on disk.

Initially developed in a cooperation between INRIA and CISCO, NDPMOn was packaged and distributed under an Open Source license with support from EMANICS. The component is now available at the following URL : http://ndpmon.sourceforge.net. The packaging helped us to gain a new community of contributors. A direct impact from this is the availability of new packages for Debian and Ubuntu distributions.

4 Summary and Conclusions

The activity in WP6 started immediately after the kickoff meeting in late January 2006. A second call was made in march 2007 for additional software development initiatives. Two tasks were launched in this work-package, one on open source packages inventory and one in the support of EMANICS-brewed Open Source packages. For both tasks the progress is measurable and concrete results are available to the public : online inventory database and 6 EMANICS Open Source projects which all do already provide to the public the enhancements enabled by the network. In summary, all phase 1 objectives have been met.

Phase 2 will continue the software support activity. The inventory repository & database activity will disappear since the infrastructure has been setup in phase 1. Finally a new activity dedicated to 3rd party Open Source software packaging is open for support in the network.

5 References

[1]The EMANICS Web site. http://www. EMANICS.org

[2]The Simple-Web. http://www.simple-web.org

[3]Libre Source. http://libresource.inria.fr

[4]B.Stroustrup, "The C++ Programming Language," Addison-Wesley, Reading, MA, 1986.

[5]B.W.Kernigham, D.M.Ritchie, "The C Programming Language," Prentice-Hall, New Jersey, 1978.

[6]K.McCarthy, G.Pavlou, S.Bhatti, J.Neuman De Souza, "Exploiting the Power of OSI Management for the Control of SNMP-capable Resources Using Generic Application Level Gateways," ISINM'95.

[7]ITU-T X.500, Information Processing - Open Systems Interconnection - The Directory: Overview of Concepts, Models and Service, 1988.

[8]ITU-T X.701, Information Technology - Open Systems Interconnection - Systems Management Overview, July 1991.

[9]ITU-T X.710, Information Technology - Open Systems Interconnection - Common Management Information Service Definition, Version 2, July 1991.

[10] ITU-T X.711, Information Technology - Open Systems Interconnection - Common Management Information Protocol Specification, Version 2, 7/91.

[11] T.Tin, G.Pavlou, R.Shi, "Tcl-MCMIS: Interpreted Management Access Facilities," Proc. of the 6th IFIP/IEEE International Workshop on Distributed Systems: Operations & Management, Ottawa, Canada, October 1995.

[12] ITU-T X.721, Information Technology - Open Systems Interconnection - Structure of Management Information - Part 2: Definition of Management Information, February 1992.

[13] ITU-T X.722, Information Technology - Open Systems Interconnection-Structure of Management Information - Part 4: Guidelines for the Definition of Managed Objects, August 1991.

[14] ITU-T X.730, Information Technology - Open Systems Interconnection - Systems Management: Object Management Function, January 1992.

[15] ITU-T X.733, Information Technology - Open Systems Interconnection - Systems Management: Alarm Reporting Function, February 1992.

[16] ITU-T X.734, Information Technology - Open Systems Interconnection - Systems Management: Event Management Function, February 1992.

[17] ITU-T X.735, Information Technology - Open Systems Interconnection - Systems Management: Log Control Function, September 1992.

[18] ITU-T X.738, Information Technology - Open Systems Interconnection - Systems Management: Metric Objects and Attributes, 1994.

[19] ITU-T X.739, Information Technology - Open Systems Interconnection - Systems Management: Summarisation Function, 1994.

[20] ITU-T X.741, Information Technology - Open Systems Interconnection - Systems Management: Objects and attributes for access control, 1995.

[21] M.T. Rose, J.P. Onions, C.J. Robbins, “The ISO Development Environment User’s Manual Version 7.0” PSI Inc / X-Tel Services Ltd., July 1991.

[22] M. Burgess, "A Site Configuration Engine", Computing Systems, 8(3), 1995

[23] J. Schoenwaelder, "Specific Simple Network Management Tools", Proc. LISA XV, December 2001

[24] IBM Corporation, General Programming Concepts: Writing and Debugging Programs, Available at http://www.unet.univie.ac.at/aix/aixprggd/genprogc/toc.htm, September 1999.

Abbreviations

AAA	Authentication, Authorization and Accounting
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
ASN.1	Abstract Syntax Notation Number One
BEEP	Bloc Extensible Exchange Protocol
BSD	Berkeley software distribution
CSS	Cascading Style Sheets
CMIP	Common Management Information Protocol
CMIS	Common Management Information Service
CMS	Content Management System
CRLF	Carriage Return and Line Feed
DMI	Definition of Management Information
DOM	Document Object Model
GDMO	Guidelines for the Definition of Managed Objects
GMS	Generic Management System
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secured
ISO	International Standards Organization
ITU	International Telecommunications Union
LGPL	Lesser general Public License
MIB	Management Information Base
MOME	Monitoring and Measurment Cluster
NoE	Network of Excellence
OSI	Open Systems Inteconnection
SMC	Self Managed Cell
SMF	System Management Function
SMI	Structure of Management Information
SNMP	Simple Network Management Protocol
SQL	Structured Query Language
SSH	Secure Shell

URL	Uniform Resource Locator
VLAN	Virtual Local Area Network
XML	eXtensible Markup Language
XSLT	eXtensible Stylesheet Language Transformation

6 Acknowledgement

This deliverable was made possible due to the large and open help of the Work Package 6 team of the EMANICS NoE. Many thanks to all of them.

7 Appendix A : Form for the calls for Open Source development support

EMANICS Work Package 6 : Open Source Software Initiatives
Proposal Sheet (to be filled & sent to the WP Leader : Olivier Festor + on WP6 mailing list before march 12, 2006)

1. Overall Open Source Software Description
(A 1/2 to 3/4 page description of the concerned software, its current status, its visbility, its use, ... In case of a non existing soft the emphasis should be made on its need and its impact on the community)

2. Licensing & distribution scheme
(license type & distribution scheme used for the software : GPL, LGPL, QPL, ...; available on a given forge, is it or will it be embedded in a third party software distribution, ..., ...)

3. Detailed List of Extensions Planed under the support of EMANICS
(describe all tasks planed and extensions envisioned as part of this support)

4. Expected Impact
(what new "markets" the enhanced software will conquer, how many distributions are envisioned, where is it going to be integrated, what visibility the NoE can gain through this support ?)

5. Cooperation level
(which parts of the extensions planed come from a cooperation among one or more partners in EMANICS, e.g. X will integrate in his software the algorithm defined by Y).

6. Cost & Requested support
(Expected cost overall + requested support. Cost includes the ressources the partner puts on the development without being supported by the NoE. These efforts must be mesurable at the end of the funding period. Request Support contains the amount of money asked to the NoE. The requested support should precisely specify how it is distributed among salary & equipment.)

8 Appendix B : SCLI cfengine integration

8.1 SCLI protocol ABNF specification

```
MESSAGE   = REQUEST / RESPONSE

REQUEST   = MREQUEST / SREQUEST
MREQUEST  = TOKEN *(WSP TOKEN) CR LF
SREQUEST  = INTERP WSP MREQUEST
TOKEN     = DQTOKEN / SQTOKEN / NQTOKEN
DQTOKEN   = DQUOTE NQTOKEN DQUOTE
SQTOKEN   = SQUOTE NQTOKEN SQUOTE
NQTOKEN   = 1*REQCHAR

RESPONSE  = MRESPONSE WSP [SRESPONSE]
MRESPONSE = *(CODE "-" DATA CR LF) CODE WSP DATA CR LF
SRESPONSE = *(INTERP WSP CODE "-" DATA CR LF) INTERP WSP CODE WSP DATA
CR LF
DATA      = *RESCHAR

CODE      =  "100"      ;arbitrary message
CODE      =/ "200"      ;normal return code
CODE      =/ "201"      ;return and exit the command loop
CODE      =/ "300"      ;generic error return code
CODE      =/ "301"      ;no association to SNMP peer
CODE      =/ "302"      ;command does not support XML
CODE      =/ "400"      ;generic syntax error
CODE      =/ "401"      ;syntax error in number of args
CODE      =/ "402"      ;syntax error in regexp
CODE      =/ "403"      ;syntax error in number
CODE      =/ "404"      ;syntax error in value
CODE      =/ "405"      ;syntax error in tokenizer
CODE      =/ "406"      ;syntax error unknown command
CODE      =/ "500"      ;snmp error return code
CODE      =/ "501"      ;snmp name lookup error

INTERP    = ALPHA *(ALPHA / DIGIT / %x2E / %x3A)
                ;Combination of letters, digits, dots and colon
                ;starting with a letter
```

```
REQCHAR    = WSP / %x21 / %x24-26 / %x28-7E
```

8.2 Example Communication Using scli Protocol Mode

```
S: 100 scli version 0.3.1 (c) 2001-2007 Juergen Schoenwaelder
C: set scli mode protocol
S: 200 ok; scli 0.3.1 ready
C: show system info
S: 301 no association to a remote SNMP agent
C: open curve
S: 200 ok; scli 0.3.1 ready
C: show system info
S: 200-Name:             curve
S: 200-Agent:            snmp://public@192.168.1.33:161//
S: 200-Description:           ProCurve J4903A Switch 2824, revision
I.08.105, ROM
S: 200-                  I.08.07 (/sw/code/build/mako(ts_08_5))
S: 200-Contact:          <j.schoenwaelder@iu-bremen.de>
S: 200-Location:         Jacobs University Bremen, Germany
S: 200-Vendor:           Hewlett Packard <http://www.hp.com/>
S: 200-Services:         datalink transport application
S: 200-Agent-Boot-Time:  2007-01-29 00:49:09 +01:00
S: 200-Interfaces:       39
S: 200-Interface Swap:   2007-03-28 11:39:15 +02:00
S: 200-Entity Swap:      2007-01-29 00:49:09 +01:00
S: 200-Bridge Ports:     24
S: 200-Bridge Type:      transparent-only
S: 200 ok; scli 0.3.1 ready
C: show system info foo
S: 401 wrong number of arguments: should be `show system info'
C: set scli timeout 100
S: 200 ok; scli 0.3.1 ready
C: set scli timeout foo
S: 403 invalid number "foo"
C: set scli "timeout" 100
S: 200 ok; scli 0.3.1 ready
C: set scli "timeout 50
```

```
S: 405 failed to tokenize input: Text ended before matching quote was
found for ". (The text was 'set scli "timeout 50')
C: set system name foo
S: 500 noAccess @ varbind 0
C: foo
S: 406 invalid command name "foo"
C: exit
S: 201 ok; scli 0.3.1 exiting
```

8.3 *Example Communication Using scli Protocol Mode in a Slave Interpreter*

```
S: 100 scli version 0.3.1 (c) 2001-2007 Juergen Schoenwaelder
C: create scli interp foo
S: 200 ok; scli 0.3.1 ready
C: foo show scli info
S: foo 200-Version:           0.3.1
S: foo 200-Epoch:             1175195954
S: foo 200-Format:            scli
S: foo 200-Interactive:       false
S: foo 200-Delay:             0 seconds
S: foo 200-Regex:             extended
S: foo 200-Debugging:
S: foo 200-Rows:              32
S: foo 200-Columns:           80
S: foo 200-Pager:             scli
S: foo 200 ok; scli 0.3.1 ready
S: 200 ok; scli 0.3.1 ready
C: foo gaga
S: foo 406 invalid command name "gaga"
S: 200 ok; scli 0.3.1 ready
C: foo set scli timeout 400
S: foo 200 ok; scli 0.3.1 ready
S: 200 ok; scli 0.3.1 ready
C: foo set scli timeout oops
S: foo 403 invalid number "oops"
S: 200 ok; scli 0.3.1 ready
C: exit
```

```
S: 201 ok; scli 0.3.1 exiting
```

8.4 Partial example output of cfengine execution with scli interpreter

```
*********************************************************************
 Main Tree Sched: shellcommands pass 1 @ Thu Jul 12 15:51:51 2007
*********************************************************************



(Non privileged user...)


cfengine:enterprise:  Nothing  promised  for  [shellcommand./bin/echo
test] (0/1 minutes elapsed)
---------------------------------------------------------------------
SCLI / SNMP script phase
---------------------------------------------------------------------


cfengine:enterprise:
Constructing scli monologue open 127.0.0.1...(timeout=0,uid=-1,gid=-1)
to-scli: open 127.0.0.1
cfengine:enterprise:
Constructing scli monologue help...(timeout=0,uid=-1,gid=-1)
to-scli: help
cf:hostname: version 0.3.1 (c) 2001-2007 Juergen Schoenwaelder
cf:hostname:  Scli  is  a  command  interpreter  which  can  be  used  to
browse,
cf:hostname: monitor and configure SNMP enabled devices. All scli com-
mands
cf:hostname: are organized in a hierarchy. The top-level commands are:
cf:hostname:
cf:hostname:  - open         Establish an association to a remote SNMP
agent.
cf:hostname:  - close         Close the association to a remote SNMP
agent.
cf:hostname:  - exit         Exit the scli command interpreter.
cf:hostname:  - help         Show this help information.
cf:hostname:  - history       Show the history of the last scli com-
mands.
cf:hostname:  - create        Create object instances on the remote
SNMP agent.
```

```
cf:hostname:   - delete        Delete object instances from the remote
SNMP agent.
cf:hostname:   - set            Modify object instances on the remote
SNMP agent.
cf:hostname:   - show          Show information provided by the remote
SNMP agent.
cf:hostname:   - monitor       Monitor information provided by the re-
mote SNMP agent.
cf:hostname:   - dump           Dump scli command sequences to restore
configurations.
cf:hostname:
cf:hostname: Use the "show scli command tree" command to browse the
complete
cf:hostname: scli command tree and the "show scli modes" command to
obtain
cf:hostname: a detailed description of the various scli commands.
cf:hostname: ok; scli 0.3.1 ready
cfengine:enterprise: Nothing promised for [scli./usr/local/bin/scli]
(0/1 minutes elapsed)
```

9 Appendix C : A cfengine execution sample with built-in patterns

```
cfengine:node1:/bin/echo $(hos: node1
cfengine:node1:/bin/echo $(hos: node1
cfengine:node1:/bin/echo $(hos: node1
cfengine:node1:/bin/echo $(hos: node1
cfengine:node1:/bin/cat /tmp/e: cfrun(0):        .......... [ Hailing node2 ]
..........
cfengine:node1:/bin/cat /tmp/e: - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - -
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:
cfengine:node1:/bin/cat /tmp/e: Executing script /bin/echo $(host-
name)...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/echo $(hos: node2
cfengine:node1:/bin/cat /tmp/e: cfengine:node2: Finished script /bin/echo $(hostname)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:
cfengine:node1:/bin/cat /tmp/e: Executing script /usr/local/sbin/cfrun node5 2>&1 >
/tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2: Finished script /usr/local/sbin/cfrun
node5 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:
cfengine:node1:/bin/cat /tmp/e: Executing script /usr/local/sbin/cfrun node6 2>&1 >
/tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2: Finished script /usr/local/sbin/cfrun
node6 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:
cfengine:node1:/bin/cat /tmp/e: Executing script /bin/echo $(host-
name)...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/echo $(hos: node2
cfengine:node1:/bin/cat /tmp/e: cfengine:node2: Finished script /bin/echo $(hostname)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:
cfengine:node1:/bin/cat /tmp/e: Executing script /usr/local/sbin/cfrun node5 2>&1 >
/tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2: Finished script /usr/local/sbin/cfrun
node5 2>&1 > /tmp/echorun.$$
```

```
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:
cfengine:node1:/bin/cat /tmp/e: Executing script /usr/local/sbin/cfrun node6 2>&1 >
/tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2: Finished script /usr/local/sbin/cfrun
node6 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:
cfengine:node1:/bin/cat /tmp/e: Executing script /usr/local/sbin/cfrun node7 2>&1 >
/tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2: Finished script /usr/local/sbin/cfrun
node7 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:
cfengine:node1:/bin/cat /tmp/e: Executing script /bin/echo $(host-
name)...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/echo $(hos: node2
cfengine:node1:/bin/cat /tmp/e: cfengine:node2: Finished script /bin/echo $(hostname)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:
cfengine:node1:/bin/cat /tmp/e: Executing script /usr/local/sbin/cfrun node5 2>&1 >
/tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2: Finished script /usr/local/sbin/cfrun
node5 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:
cfengine:node1:/bin/cat /tmp/e: Executing script /bin/echo $(host-
name)...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/echo $(hos: node2
cfengine:node1:/bin/cat /tmp/e: cfengine:node2: Finished script /bin/echo $(hostname)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:
cfengine:node1:/bin/cat /tmp/e: Executing script /usr/local/sbin/cfrun node5 2>&1 >
/tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2: Finished script /usr/local/sbin/cfrun
node5 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:
cfengine:node1:/bin/cat /tmp/e: Executing script /usr/local/sbin/cfrun node6 2>&1 >
/tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2: Finished script /usr/local/sbin/cfrun
node6 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:
cfengine:node1:/bin/cat /tmp/e: Executing script /usr/local/sbin/cfrun node7 2>&1 >
/tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2: Finished script /usr/local/sbin/cfrun
node7 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:
cfengine:node1:/bin/cat /tmp/e: Executing script /usr/bin/pgrep cfrun > /dev/null;
while [ $? = 0 ]; do pgrep cfrun > /dev/null; done...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2: Finished script /usr/bin/pgrep cfrun >
/dev/null; while [ $? = 0 ]; do pgrep cfrun > /dev/null; done
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:
cfengine:node1:/bin/cat /tmp/e: Executing script /bin/cat
/tmp/echorun.*...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfrun(0):
.......... [ Hailing node5 ] ..........
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - - - - - - -
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfengine:node5:
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: Executing script
/bin/echo $(hostname)...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfen-
gine:node5:/bin/echo $(hos: node5
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfengine:node5: Finis-
hed script /bin/echo $(hostname)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfengine:node5:
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: Executing script
/usr/local/sbin/cfrun node14 2>&1 > /tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfengine:node5: Finis-
hed script /usr/local/sbin/cfrun node14 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfengine:node5:
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: Executing script
/bin/echo $(hostname)...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfen-
gine:node5:/bin/echo $(hos: node5
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfengine:node5: Finis-
hed script /bin/echo $(hostname)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfengine:node5:
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: Executing script
/usr/local/sbin/cfrun node14 2>&1 > /tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfengine:node5: Finis-
hed script /usr/local/sbin/cfrun node14 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfengine:node5:
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: Executing script
/usr/local/sbin/cfrun node15 2>&1 > /tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfengine:node5: Finis-
hed script /usr/local/sbin/cfrun node15 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfengine:node5:
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: Executing script
/usr/local/sbin/cfrun node16 2>&1 > /tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfengine:node5: Finis-
hed script /usr/local/sbin/cfrun node16 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfengine:node5:
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: Executing script
/bin/echo $(hostname)...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfen-
gine:node5:/bin/echo $(hos: node5
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfengine:node5: Finis-
hed script /bin/echo $(hostname)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfengine:node5:
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: Executing script
/usr/local/sbin/cfrun node14 2>&1 > /tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfengine:node5: Finis-
hed script /usr/local/sbin/cfrun node14 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfengine:node5:
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: Executing script
/usr/local/sbin/cfrun node15 2>&1 > /tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfengine:node5: Finis-
hed script /usr/local/sbin/cfrun node15 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfengine:node5:
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: Executing script
/bin/echo $(hostname)...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfen-
gine:node5:/bin/echo $(hos: node5
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfengine:node5: Finis-
hed script /bin/echo $(hostname)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfengine:node5:
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: Executing script
/usr/local/sbin/cfrun node14 2>&1 > /tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfengine:node5: Finis-
hed script /usr/local/sbin/cfrun node14 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfengine:node5:
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: Executing script
/usr/local/sbin/cfrun node15 2>&1 > /tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfengine:node5: Finis-
hed script /usr/local/sbin/cfrun node15 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfengine:node5:
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: Executing script
/usr/local/sbin/cfrun node16 2>&1 > /tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfengine:node5: Finis-
hed script /usr/local/sbin/cfrun node16 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfengine:node5:
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: Executing script
/usr/bin/pgrep cfrun > /dev/null; while [ $? = 0 ]; do pgrep cfrun > /dev/null;
done...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfengine:node5: Finis-
hed script /usr/bin/pgrep cfrun > /dev/null; while [ $? = 0 ]; do pgrep cfrun >
/dev/null; done
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfengine:node5:
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: Executing script
/bin/cat /tmp/echorun.*...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfen-
gine:node5:/bin/cat /tmp/e: cfrun(0):         .......... [ Hailing node14 ] ..........
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfen-
```

```
gine:node5:/bin/cat /tmp/e: - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - -
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfen-
gine:node5:/bin/cat /tmp/e: cfengine:node14:
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfen-
gine:node5:/bin/cat /tmp/e: Executing script /bin/echo $(hostname)...(timeout=0,uid=-
1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfen-
gine:node5:/bin/cat /tmp/e: cfengine:node14:/bin/echo $(hos: node14
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfen-
gine:node5:/bin/cat /tmp/e: cfengine:node14: Finished script /bin/echo $(hostname)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfen-
gine:node5:/bin/cat /tmp/e: - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - -
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfen-
gine:node5:/bin/cat /tmp/e: cfrun(0):         .......... [ Hailing node15 ] ..........
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfen-
gine:node5:/bin/cat /tmp/e: - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - -
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfen-
gine:node5:/bin/cat /tmp/e: cfengine:node15:
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfen-
gine:node5:/bin/cat /tmp/e: Executing script /bin/echo $(hostname)...(timeout=0,uid=-
1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfen-
gine:node5:/bin/cat /tmp/e: cfengine:node15:/bin/echo $(hos: node15
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfen-
gine:node5:/bin/cat /tmp/e: cfengine:node15: Finished script /bin/echo $(hostname)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfen-
gine:node5:/bin/cat /tmp/e: - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - -
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfen-
gine:node5:/bin/cat /tmp/e: cfrun(0):         .......... [ Hailing node16 ] ..........
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfen-
gine:node5:/bin/cat /tmp/e: - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - -
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfen-
gine:node5:/bin/cat /tmp/e: cfengine:node16:
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfen-
gine:node5:/bin/cat /tmp/e: Executing script /bin/echo $(hostname)...(timeout=0,uid=-
1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfen-
gine:node5:/bin/cat /tmp/e: cfengine:node16:/bin/echo $(hos: node16
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfen-
gine:node5:/bin/cat /tmp/e: cfengine:node16: Finished script /bin/echo $(hostname)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfen-
gine:node5:/bin/cat /tmp/e: - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - -
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfengine:node5: Finis-
hed script /bin/cat /tmp/echorun.*
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfengine:node5: Dele-
ting file /tmp/echorun.31035
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfengine:node5: Dele-
ting file /tmp/echorun.31041
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfengine:node5: Dele-
ting file /tmp/echorun.31038
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfrun(0):
.......... [ Hailing node6 ] ..........
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfengine:node6:
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: Executing script
/bin/echo $(hostname)...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfen-
gine:node6:/bin/echo $(hos: node6
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfengine:node6: Finis-
hed script /bin/echo $(hostname)
```

```
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - -
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfrun(0):
.......... [ Hailing node7 ] ..........
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - -
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfengine:node7:
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: Executing script
/bin/echo $(hostname)...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfen-
gine:node7:/bin/echo $(hos: node7
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: cfengine:node7: Finis-
hed script /bin/echo $(hostname)
cfengine:node1:/bin/cat /tmp/e: cfengine:node2:/bin/cat /tmp/e: - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - -
cfengine:node1:/bin/cat /tmp/e: cfengine:node2: Finished script /bin/cat
/tmp/echorun.*
cfengine:node1:/bin/cat /tmp/e: cfengine:node2: Deleting file /tmp/echorun.3309
cfengine:node1:/bin/cat /tmp/e: cfengine:node2: Deleting file /tmp/echorun.3313
cfengine:node1:/bin/cat /tmp/e: cfengine:node2: Deleting file /tmp/echorun.3316
cfengine:node1:/bin/cat /tmp/e: - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - -
cfengine:node1:/bin/cat /tmp/e: cfrun(0):         .......... [ Hailing node3 ]
..........
cfengine:node1:/bin/cat /tmp/e: - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - -
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:
cfengine:node1:/bin/cat /tmp/e: Executing script /bin/echo $(host-
name)...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/echo $(hos: node3
cfengine:node1:/bin/cat /tmp/e: cfengine:node3: Finished script /bin/echo $(hostname)
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:
cfengine:node1:/bin/cat /tmp/e: Executing script /usr/local/sbin/cfrun node8 2>&1 >
/tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node3: Finished script /usr/local/sbin/cfrun
node8 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:
cfengine:node1:/bin/cat /tmp/e: Executing script /usr/local/sbin/cfrun node9 2>&1 >
/tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node3: Finished script /usr/local/sbin/cfrun
node9 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:
cfengine:node1:/bin/cat /tmp/e: Executing script /bin/echo $(host-
name)...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/echo $(hos: node3
cfengine:node1:/bin/cat /tmp/e: cfengine:node3: Finished script /bin/echo $(hostname)
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:
cfengine:node1:/bin/cat /tmp/e: Executing script /usr/local/sbin/cfrun node8 2>&1 >
/tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node3: Finished script /usr/local/sbin/cfrun
node8 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:
cfengine:node1:/bin/cat /tmp/e: Executing script /usr/local/sbin/cfrun node9 2>&1 >
/tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node3: Finished script /usr/local/sbin/cfrun
node9 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:
cfengine:node1:/bin/cat /tmp/e: Executing script /usr/local/sbin/cfrun node10 2>&1 >
/tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node3: Finished script /usr/local/sbin/cfrun
node10 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:
cfengine:node1:/bin/cat /tmp/e: Executing script /bin/echo $(host-
name)...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/echo $(hos: node3
cfengine:node1:/bin/cat /tmp/e: cfengine:node3: Finished script /bin/echo $(hostname)
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:
cfengine:node1:/bin/cat /tmp/e: Executing script /usr/local/sbin/cfrun node8 2>&1 >
/tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
```

```
cfengine:node1:/bin/cat /tmp/e: cfengine:node3: Finished script /usr/local/sbin/cfrun
node8 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:
cfengine:node1:/bin/cat /tmp/e: Executing script /bin/echo $(host-
name)...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/echo $(hos: node3
cfengine:node1:/bin/cat /tmp/e: cfengine:node3: Finished script /bin/echo $(hostname)
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:
cfengine:node1:/bin/cat /tmp/e: Executing script /usr/local/sbin/cfrun node8 2>&1 >
/tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node3: Finished script /usr/local/sbin/cfrun
node8 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:
cfengine:node1:/bin/cat /tmp/e: Executing script /usr/local/sbin/cfrun node9 2>&1 >
/tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node3: Finished script /usr/local/sbin/cfrun
node9 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:
cfengine:node1:/bin/cat /tmp/e: Executing script /usr/local/sbin/cfrun node10 2>&1 >
/tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node3: Finished script /usr/local/sbin/cfrun
node10 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:
cfengine:node1:/bin/cat /tmp/e: Executing script /usr/bin/pgrep cfrun > /dev/null;
while [ $? = 0 ]; do pgrep cfrun > /dev/null; done...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node3: Finished script /usr/bin/pgrep cfrun >
/dev/null; while [ $? = 0 ]; do pgrep cfrun > /dev/null; done
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:
cfengine:node1:/bin/cat /tmp/e: Executing script /bin/cat
/tmp/echorun.*...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfrun(0):
.......... [ Hailing node8 ] ..........
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfengine:node8:
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: Executing script
/bin/echo $(hostname)...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfen-
gine:node8:/bin/echo $(hos: node8
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfengine:node8: Finis-
hed script /bin/echo $(hostname)
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfengine:node8:
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: Executing script
/usr/local/sbin/cfrun node17 2>&1 > /tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfengine:node8: Finis-
hed script /usr/local/sbin/cfrun node17 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfengine:node8:
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: Executing script
/bin/echo $(hostname)...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfen-
gine:node8:/bin/echo $(hos: node8
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfengine:node8: Finis-
hed script /bin/echo $(hostname)
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfengine:node8:
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: Executing script
/usr/local/sbin/cfrun node17 2>&1 > /tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfengine:node8: Finis-
hed script /usr/local/sbin/cfrun node17 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfengine:node8:
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: Executing script
/usr/local/sbin/cfrun node18 2>&1 > /tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfengine:node8: Finis-
hed script /usr/local/sbin/cfrun node18 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfengine:node8:
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: Executing script
/bin/echo $(hostname)...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfen-
gine:node8:/bin/echo $(hos: node8
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfengine:node8: Finis-
```

```
hed script /bin/echo $(hostname)
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfengine:node8:
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: Executing script
/usr/local/sbin/cfrun node17 2>&1 > /tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfengine:node8: Finis-
hed script /usr/local/sbin/cfrun node17 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfengine:node8:
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: Executing script
/usr/local/sbin/cfrun node18 2>&1 > /tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfengine:node8: Finis-
hed script /usr/local/sbin/cfrun node18 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfengine:node8:
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: Executing script
/usr/local/sbin/cfrun node19 2>&1 > /tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfengine:node8: Finis-
hed script /usr/local/sbin/cfrun node19 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfengine:node8:
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: Executing script
/bin/echo $(hostname)...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfen-
gine:node8:/bin/echo $(hos: node8
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfengine:node8: Finis-
hed script /bin/echo $(hostname)
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfengine:node8:
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: Executing script
/usr/local/sbin/cfrun node17 2>&1 > /tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfengine:node8: Finis-
hed script /usr/local/sbin/cfrun node17 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfengine:node8:
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: Executing script
/usr/local/sbin/cfrun node18 2>&1 > /tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfengine:node8: Finis-
hed script /usr/local/sbin/cfrun node18 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfengine:node8:
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: Executing script
/usr/local/sbin/cfrun node19 2>&1 > /tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfengine:node8: Finis-
hed script /usr/local/sbin/cfrun node19 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfengine:node8:
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: Executing script
/usr/bin/pgrep cfrun > /dev/null; while [ $? = 0 ]; do pgrep cfrun > /dev/null;
done...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfengine:node8: Finis-
hed script /usr/bin/pgrep cfrun > /dev/null; while [ $? = 0 ]; do pgrep cfrun >
/dev/null; done
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfengine:node8:
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: Executing script
/bin/cat /tmp/echorun.*...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfen-
gine:node8:/bin/cat /tmp/e: cfrun(0):         .......... [ Hailing node17 ] ..........
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfen-
gine:node8:/bin/cat /tmp/e: - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - -
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfen-
gine:node8:/bin/cat /tmp/e: cfengine:node17:
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfen-
gine:node8:/bin/cat /tmp/e: Executing script /bin/echo $(hostname)...(timeout=0,uid=-
1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfen-
gine:node8:/bin/cat /tmp/e: cfengine:node17:/bin/echo $(hos: node17
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfen-
gine:node8:/bin/cat /tmp/e: cfengine:node17: Finished script /bin/echo $(hostname)
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfen-
gine:node8:/bin/cat /tmp/e: - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - -
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfen-
gine:node8:/bin/cat /tmp/e: cfrun(0):         .......... [ Hailing node18 ] ..........
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfen-
gine:node8:/bin/cat /tmp/e: - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```
- - - - - - - - - - - -
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfen-
gine:node8:/bin/cat /tmp/e: cfengine:node18:
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfen-
gine:node8:/bin/cat /tmp/e: Executing script /bin/echo $(hostname)...(timeout=0,uid=-
1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfen-
gine:node8:/bin/cat /tmp/e: cfengine:node18:/bin/echo $(hos: node18
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfen-
gine:node8:/bin/cat /tmp/e: cfengine:node18: Finished script /bin/echo $(hostname)
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfen-
gine:node8:/bin/cat /tmp/e: - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - -
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfen-
gine:node8:/bin/cat /tmp/e: cfrun(0):         .......... [ Hailing node19 ] ..........
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfen-
gine:node8:/bin/cat /tmp/e: - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - -
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfen-
gine:node8:/bin/cat /tmp/e: cfengine:node19:
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfen-
gine:node8:/bin/cat /tmp/e: Executing script /bin/echo $(hostname)...(timeout=0,uid=-
1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfen-
gine:node8:/bin/cat /tmp/e: cfengine:node19:/bin/echo $(hos: node19
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfen-
gine:node8:/bin/cat /tmp/e: cfengine:node19: Finished script /bin/echo $(hostname)
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfen-
gine:node8:/bin/cat /tmp/e: - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - -
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfengine:node8: Finis-
hed script /bin/cat /tmp/echorun.*
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfengine:node8: Dele-
ting file /tmp/echorun.31911
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfengine:node8: Dele-
ting file /tmp/echorun.31915
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfengine:node8: Dele-
ting file /tmp/echorun.31918
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - -
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfrun(0):
.......... [ Hailing node9 ] ..........
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - -
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfengine:node9:
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: Executing script
/bin/echo $(hostname)...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfen-
gine:node9:/bin/echo $(hos: node9
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfengine:node9: Finis-
hed script /bin/echo $(hostname)
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - -
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfrun(0):
.......... [ Hailing node10 ] ..........
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - -
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfengine:node10:
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: Executing script
/bin/echo $(hostname)...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfen-
gine:node10:/bin/echo $(hos: node10
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: cfengine:node10: Fi-
nished script /bin/echo $(hostname)
cfengine:node1:/bin/cat /tmp/e: cfengine:node3:/bin/cat /tmp/e: - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - -
cfengine:node1:/bin/cat /tmp/e: cfengine:node3: Finished script /bin/cat
/tmp/echorun.*
cfengine:node1:/bin/cat /tmp/e: cfengine:node3: Deleting file /tmp/echorun.31043
```

```
cfengine:node1:/bin/cat /tmp/e: cfengine:node3: Deleting file /tmp/echorun.31047
cfengine:node1:/bin/cat /tmp/e: cfengine:node3: Deleting file /tmp/echorun.31050
cfengine:node1:/bin/cat /tmp/e: - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - -
cfengine:node1:/bin/cat /tmp/e: cfrun(0):        .......... [ Hailing node4 ]
..........
cfengine:node1:/bin/cat /tmp/e: - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - -
cfengine:node1:/bin/cat /tmp/e: cfengine:node4:
cfengine:node1:/bin/cat /tmp/e: Executing script /bin/echo $(host-
name)...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node4:/bin/echo $(hos: node4
cfengine:node1:/bin/cat /tmp/e: cfengine:node4: Finished script /bin/echo $(hostname)
cfengine:node1:/bin/cat /tmp/e: cfengine:node4:
cfengine:node1:/bin/cat /tmp/e: Executing script /usr/local/sbin/cfrun node11 2>&1 >
/tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node4: Finished script /usr/local/sbin/cfrun
node11 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node4:
cfengine:node1:/bin/cat /tmp/e: Executing script /usr/local/sbin/cfrun node12 2>&1 >
/tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node4: Finished script /usr/local/sbin/cfrun
node12 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node4:
cfengine:node1:/bin/cat /tmp/e: Executing script /usr/local/sbin/cfrun node13 2>&1 >
/tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node4: Finished script /usr/local/sbin/cfrun
node13 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node4:
cfengine:node1:/bin/cat /tmp/e: Executing script /bin/cat
/tmp/echorun.*...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node4: Finished script /bin/cat
/tmp/echorun.*
cfengine:node1:/bin/cat /tmp/e: cfengine:node4: Deleting file /tmp/echorun.849
cfengine:node1:/bin/cat /tmp/e: cfengine:node4: Deleting file /tmp/echorun.852
cfengine:node1:/bin/cat /tmp/e: cfengine:node4:
cfengine:node1:/bin/cat /tmp/e: Executing script /bin/echo $(host-
name)...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node4:/bin/echo $(hos: node4
cfengine:node1:/bin/cat /tmp/e: cfengine:node4: Finished script /bin/echo $(hostname)
cfengine:node1:/bin/cat /tmp/e: cfengine:node4:
cfengine:node1:/bin/cat /tmp/e: Executing script /usr/local/sbin/cfrun node11 2>&1 >
/tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node4: Finished script /usr/local/sbin/cfrun
node11 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node4:
cfengine:node1:/bin/cat /tmp/e: Executing script /usr/local/sbin/cfrun node12 2>&1 >
/tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node4: Finished script /usr/local/sbin/cfrun
node12 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node4:
cfengine:node1:/bin/cat /tmp/e: Executing script /usr/local/sbin/cfrun node13 2>&1 >
/tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node4: Finished script /usr/local/sbin/cfrun
node13 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node4:
cfengine:node1:/bin/cat /tmp/e: Executing script /bin/echo $(host-
name)...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node4:/bin/echo $(hos: node4
cfengine:node1:/bin/cat /tmp/e: cfengine:node4: Finished script /bin/echo $(hostname)
cfengine:node1:/bin/cat /tmp/e: cfengine:node4:
cfengine:node1:/bin/cat /tmp/e: Executing script /usr/local/sbin/cfrun node11 2>&1 >
/tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node4: Finished script /usr/local/sbin/cfrun
node11 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node4:
cfengine:node1:/bin/cat /tmp/e: Executing script /usr/local/sbin/cfrun node12 2>&1 >
/tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node4: Finished script /usr/local/sbin/cfrun
```

```
node12 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: cfengine:node4:
cfengine:node1:/bin/cat /tmp/e: Executing script /bin/echo $(host-
name)...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node4:/bin/echo $(hos: node4
cfengine:node1:/bin/cat /tmp/e: cfengine:node4: Finished script /bin/echo $(hostname)
cfengine:node1:/bin/cat /tmp/e: cfengine:node4:
cfengine:node1:/bin/cat /tmp/e: Executing script /usr/local/sbin/cfrun node11 2>&1 >
/tmp/echorun.$$...(timeout=0,uid=-1,gid=-1)
cfengine:node1:/bin/cat /tmp/e: cfengine:node4: Finished script /usr/local/sbin/cfrun
node11 2>&1 > /tmp/echorun.$$
cfengine:node1:/bin/cat /tmp/e: - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - -
```